HOKKAIDO UNIVERSITY

DOCTORAL THESIS

A Robust Self-Constructing Normalized Gaussian Network for Online Machine Learning

Author:
Jana Cathrin BACKHUS

Supervisor:
Prof. Masanori SUGIMOTO

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Laboratory of Intelligent Information Systems
Division of Computer Science and Information Technology

February 14, 2017

Declaration of Authorship

I, Jana Cathrin BACKHUS, declare that this thesis titled, "A Robust Self-Constructing Normalized Gaussian Network for Online Machine Learning" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

HOKKAIDO UNIVERSITY

Abstract

Graduate School of Information Science and Technology
Division of Computer Science and Information Technology

Doctor of Philosophy

A Robust Self-Constructing Normalized Gaussian Network for Online Machine Learning

by Jana Cathrin BACKHUS

In this thesis, I aim to improve the robustness and applicability of Normalized Gaussian networks (NGnet) in the context of online machine learning tasks. A challenging problem in online machine learning is the limited domain knowledge provoked by restricted prior knowledge, while additional information are received only sequentially over time. The limited domain knowledge makes the application of artificial neural networks (ANN) more difficult, and major challenges include negative interference and the selection of an accurate model complexity. In this thesis, I consider these challenges in regard to the NGnet, which belongs to a group of ANNs that possess a certain grade of robustness against negative interference due to the local properties of their network architecture. Yet, further improvements of robustness are necessary in regard to the ANN's learning algorithm and model complexity selection. A recently proposed learning algorithm with localized forgetting provides robustness against negative interference, but it is not applicable over the full numerical range of an implied discount factor. Also, dynamic model selection was yet to be considered. Therefore, I revise the localized forgetting approach and adapt dynamic model selection to it in a self-constructing manner. Dynamic model selection has been considered for an earlier learning algorithm of the NGnet with global forgetting, which however shows a non-robust behavior in negative interference prone environments. Then, I propose localization of some of the model selection mechanisms to improve their robustness and add a new merge manipulation to deal with model redundancies. The effectiveness of the proposed method is compared with earlier learning approaches of the NGnet for several experiments. The proposed method possesses robust and favorable performance in the different tested learning environments, making it the better alternative when applied to online learning tasks with proneness to negative interference.

Acknowledgements

First of all, I would like to express my special appreciation and thanks to my advisor Professor Masanori Sugimoto, who has been a tremendous support for me, especially in the last two years after becoming my main supervisor. He was always supportive and encouraging, which helped me a lot to go on with my research even in face of hardships.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Mineichi Kudo, Prof. Hideyuki Imai, and Prof. Ichigaku Takigawa, who took their time to read my paper drafts, answer my questions and helped me to make my research better with their insightful comments and encouragement during the last few years.

My sincere thanks also goes to Prof. Hidetoshi Nonaka and Prof. Takeshi Yoshikawa, who could not walk this path to the end with me together, but they were the ones who first welcomed me to this laboratory 5 years ago and provided me with all the support that I needed to enter the degree programs and give me lots of insightful comments in my early learning stages as a researcher.

I also want to thank my fellow labmates in for the stimulating discussions, for the sleepless nights we were working together before deadlines or had a game playing session, and for all the other fun we have had in the last five years. Also, I thank my PhD. colleague Keigo Kimura, who was always ready for discussions about research or other little PhD. problems.

Last but not the least, I would like to thank my family and friends, inside and outside of Japan, for always being supportive of me and spending time with me whenever I was ready for a break. Because without them, I would not be who I am today and might never had the opportunity to study in Japan in the first place. A special thanks goes to my mum who raised me with all her might.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
BC	**B**hattacharyya **C**oefficient
CD	**C**orrelation **D**imension
CV	**C**ross-**V**alidation
EKF	**E**xtended **K**alman **F**ilter
ELM	**E**xtreme **L**earning **M**achine
FA	**F**unction **A**pproximation
GF	**G**lobal **F**orgetting
i.i.d.	**i**ndependent (and) **i**dentically **d**istributed
LF	**L**ocal **F**orgetting
MDP	**M**arkov **D**ecision **P**rocess
ME	**M**ixture (of) **E**xperts
MG	**M**ackey-**G**lass (Time Series)
ML	**M**achine **L**earning
MRAN	**M**inimum **R**esource **A**llocation **N**etwork
NGnet	**N**ormalized **G**aussian **Net**work
pdf	**p**robability **d**ensity **f**unction
RAN	**R**esource **A**llocation **N**etwork
RBFN	**R**adial **B**asis **F**unction **N**etwork
RL	**R**einforcement **L**earning
SD	**S**tandard **D**eviation

Chapter 1

Introduction

Online machine learning is one big area of machine learning, the study of algorithms that learn from and make predictions on data. For online machine learning, data samples are received sequentially over time and an applied learning system has to update its estimator in an incremental manner. This is opposing to batch learning algorithms that are applied to offline machine learning tasks and train a learning system after observing the whole training data set. A challenging concern in online machine learning is the limited domain knowledge provoked by the restricted prior knowledge about the environment and the sequential observation of additional information. In truly incremental learning schemes, one additionally assumes that training data are not only observed one-by-one but immediately processed and then discarded.

In this thesis, I mainly apply artificial neural networks (ANN) to online learning tasks. ANNs are one important group of learning systems that approximate the unknown functional relationship of an underlying learning task based on observed training data samples. In recent years, ANNs have been applied to an ever growing field of different tasks, which include time series prediction, dynamic systems, robotics and control. Yet, their application is not straight-forward and several challenges have to be resolved: the choice of a network architecture, a network model complexity and a learning algorithm (Wilamowski 2009). These selections highly influence the learning performance of the ANN. Generally, one chooses a network architecture first and then decides on the learning algorithm for training, where most of the times different learning approaches are possible for a network architecture. Afterwards, a model complexity has to be chosen, and often a trial-and-error search over different network sizes is necessary, trained and evaluated on available data samples. This model selection approach is however not applicable to online learning tasks.

Additional challenges arise for online learning tasks due to the limited domain knowledge. One big concern is to choose a model complexity that is able to represent the underlying learning problem well enough without over- or under-fitting. Here, a common approach is to select the network

model size dynamically during learning "on-the-go" by increasing or decreasing it according to some network model manipulation mechanisms. Another concern is the proneness of ANNs to negative interference when training data are not identically and independently distributed in the input space. Here, negative interference refers to the forgetting of previously learned information in favor of newly received data samples, which then leads to a decrease in learning performance. The prevention of negative interference requires the availability of either validation data, memorizing of all training data or strong prior knowledge about the learning tasks (Schaal and Atkeson 1998). These options are however not available in online learning tasks.

It is necessary to consider these concerns for a successful application of ANNs to online learning tasks. Therefore, ANNs with a robust behavior against negative interference are more commonly considered. Negative interference is less present in ANNs with local properties, and the locality of the network architecture plays an important role in regard to the robustness. ANNs with a receptive field based network architecture are then a popular choice for online learning tasks, because their architecture possesses a certain grade of local learning behavior imposed by the receptive fields. Yet, a suitable network architecture alone is not enough to mitigate negative interference, and further possibilities need to be explored to achieve robust learning behavior for ANNs. This concerns both learning algorithm and model complexity selection, since both have potential to be negatively influenced by the training data distribution and a stable learning behavior cannot be ensured without preventive measures.

1.1 Contributions

In this thesis, I focus on the Normalized Gaussian network (NGnet) (Sato and Ishii 2000) as one example of an ANN with a receptive field based network architecture. The NGnet has been considered for online machine learning tasks, and a recently proposed learning algorithm (Celaya and Agostini 2015) shows high potential to deal with negative interference prone environments by applying network updates based on localized forgetting. Then, I propose several improvements to further increase robustness and applicability of the NGnet for online machine learning tasks. My contributions are described in the following.

Revision of Update Method: I revise the previously proposed update method with localized forgetting for the NGnet, because in its proposed form it is not applicable over the full numerical range of an implied discount factor. To eliminate this disadvantage, I revise the derivation of the

localized forgetting method under the consideration of additional dependencies. The revised update method can then be applied over the full range of the implied discount factor, and performs equally well or better than the previous method as shown in the experiments.

Discussion about Update Precisions: One interesting property of updates with localized forgetting is that at each time step only a few network units are updated largely. Both update and forgetting weights become very small for units far from the currently observed data sample so that their updates are dispensable, especially outside the computational accuracies. Even within the computational accuracies, units only conduct minor updates the further they are away from the observed data sample. Therefore, I discuss the abandonment of updates with small weights under certain precision limits, since these do not affect the learning performance very much. By further decreasing the update precision even beyond computational accuracy, I achieve a reduction of computational complexity while the learning performance stays approximately the same.

Self-Constructing Dynamic Model Selection: For the localized forgetting update approach, only static model selection was previously considered. To better deal with the problem of model complexity selection in environments where domain knowledge is limited, I apply some model manipulation mechanisms that dynamically increase and decrease the model complexity during learning. These mechanisms are applied in a self-constructing manner, where the initial size of the network model is set to zero and new units are added when necessary after observing new data samples. This helps to improve the applicability of the NGnet to online learning tasks, since an initial choice of network complexity and parameters can be almost omitted before training.

Improved Robustness of Model Selection: When dealing with negative interference prone environments, it is important that the applied ANN possesses robustness against it. Furthermore, it is not enough to consider robustness only for one property of the ANN, but it should be present thoroughly for all performance influencing properties of the ANN. For the NGnet, robustness has already been considered in regard to the network architecture and the learning algorithm. Yet, it was not considered for the model complexity selection approach, which remains an open problem. The NGnet applies several dynamic model manipulation mechanisms and for some of them further improvement of robustness is possible by localizing the manipulation decision. The experimental results show also that the localized

manipulation mechanisms help to improve the learning performance. Especially, the localization of a delete mechanism shows an improved stability over all applied testbeds, which makes it the most important contribution of this thesis. In addition, I also discuss shortly why an earlier delete approach is lacking this robust performance.

Dealing with Model Redundancies In addition to the adapted model manipulation mechanisms, I propose a new merge mechanism to handle redundancies in the network model. Dealing with model redundancies has not been considered previously for the NGnet, although similar merging mechanisms have been applied to other ANNs and learning systems. Because of the computational heaviness of this approach, I propose to change its overlap calculation in order to improve the identification accuracy of redundant units when the mechanism is applied only in time intervals. The NGnet becomes then able to successfully deal with model redundancies.

1.2 Thesis Structure

The rest of the thesis is structured as follows. In Chapter 2, I give a detailed overview about different related topics, starting with some general background information about online machine learning and then discuss artificial neural networks (ANN) and its challenges when applied to the considered problem setting. Related work about ANNs with a receptive field based architecture are explored and the Normalized Gaussian network (NGnet) is introduced as one ANN, which I study more deeply in the considered context. In Chapter 3, I discuss my improvements for the NGnet that are proposed to achieve better robustness in learning and deal with the model complexity selection. Some experiments are conducted in Chapter 4 to evaluate the effectiveness of the proposed method in regard to earlier NGnet approaches for different learning tasks, including function approximation, chaotic time series approximation and reinforcement learning. In Chapter 5, I discuss the benefits and limitations of each proposed contribution based on the experimental results of Chapter 4. Finally, I draw an overall conclusion and explore possible future work in Chapter 6.

Chapter 2

Background and Related Work

In this chapter, I explore necessary background information and related work for this thesis, divided into three major sections. First, I discuss online machine learning and what challenges have to be considered (Section 2.1). Then in Section 2.2, I explore some related artificial neural networks that are applicable to online learning tasks and also provide a theoretical basis for the main focus of this thesis. Finally, I introduce the network architecture and training method for the normalized Gaussian network (NGnet) in detail in Section 2.3.

2.1 Online Machine Learning

Machine learning (ML) is the study of algorithms that are able to learn from and make predictions on data by building a model from the sample inputs. In the following, I will mainly refer to these models with *learning systems*. One big sub-area of ML is online machine learning, where sample data are observed sequentially over time, and the learning system has to be updated incrementally after observing new data to improve its prediction model. This is different from offline machine learning, where learning systems are updated in a batch mode to learn the best model from the whole training data set simultaneously. Although nowadays quite a huge amount of research concentrates on offline ML methods, almost all of the earlier works on ML have focused on online learning systems. This is mainly related to the computational simplicity of online ML updates compared with offline ones (Bottou 1998).

Incremental update approaches are preferable to batch updates in the following two cases. In non-stationary tasks, it is assumed that not all data samples are available at the begin of learning and additional data samples are observed over time that imply essentially new information for the learning system. If batch updates are applied in these cases, then the learning system would need to continuously retrain its prediction model on the whole data set each time a new data sample is observed. This implies also the memorization of all formerly observed data and becomes computationally infeasible with a growing number of data samples. On the other hand,

incremental approaches can update the learning system for each newly received data sample without retraining, eliminating also the necessity of storing all previously received data samples. Therefore, incremental learning systems are applied more naturally to non-stationary tasks. Another reason to prefer incremental update approaches is computational feasibility when the number of data samples is so large that the update of the learning system in one batch is infeasible. Then, incremental updates are necessary and learning systems are normally updated with mini-batches of data. In this thesis, I focus on the first case of non-stationary learning tasks and will put a special focus on truly incremental learning tasks.

In truly incremental learning tasks (Schaal and Atkeson 1998), it is assumed that only one data sample is observed at any time and directly discarded after learning. Because of the small number of currently accessible data samples, domain knowledge is very limited especially when no prior knowledge about the environment is available. Additionally, it is assumed that the total number of training data is unknown. Because of these limitations, it is challenging for the learning systems to learn a good prediction model, and special considerations are necessary to ensure a robust learning performance. This also includes the choice of parameters for the learning algorithm, since many learning systems are sensitive to it and a wrong choice might result in a poor learning behavior (Saad 1998). I will further discuss problems in truly incremental schemes in the context where artificial neural networks are employed as learning systems in Section 2.2.

Depending on the learning tasks, ML knows different types of feedback that can be broadly classified in three big categories of learning (Russell and Norvig 2010, Chapter 18). The three categories include unsupervised learning, supervised learning and reinforcement learning. In this thesis, I consider supervised and reinforcement learning tasks.

2.1.1 Supervised Learning

Supervised learning is often described as learning from a teacher. Learning systems receive a set of input-output training data pairs, where the output represents the target for the input and the relationship between the inputs and outputs has to be learned. Two types of supervised learning problems are usually distinguished: classification and regression. For classification, the target outputs are a finite set of values, and the goal is to learn which inputs belong to which class. In this thesis, I will however mainly consider regression tasks, where learning systems need to learn the relationship between inputs and continuous target outputs.

In regression tasks, the learning system maps the relationship as a function $f : X \rightarrow Y$ between an N-dimensional independent input $x \in \mathbb{R}^N$ and a D-dimensional input-dependent output $y \in \mathbb{R}^D$. Given T available

training samples $\{(x(t), y(t)\}_{t=1}^{T}$, the goal of the learning system is to approximate the underlying functional relationship so that it can predict the output value y for unseen values of input x. For so called *parametric* regression, one assumes that the learning system is represented by a parametrized model, for example a linear model $f(x) = Ax$. The learning task is then to find appropriate values for the model parameters A (Stulp and Sigaud 2015). In the simplest case, an appropriate representation can be inferred directly so that $y = f(x)$. For more complicated cases, it is not possible to learn the functional relationship directly for one or several of the following reasons: the number of training samples is limited, training data might contain some noise or the learning system's representation capabilities are limited. Therefore, learning systems often aim to model the predictive distribution $p(y|x)$ to deal better with the uncertainty about value y for each value x (Bishop 2006, Chapter 3). Then, the learning system tries to find a good approximation $y \approx f(x)$ by minimizing the expected value of a suitably chosen loss function. For the standard regression model, one generally assumes that target output y is given by a deterministic function $f(x)$ with additive Gaussian noise ϵ so that

$$y = f(x) + \epsilon. \tag{2.1}$$

2.1.2 Reinforcement Learning

Reinforcement learning (RL) can be described as learning from rewards or punishments that are received as feedback from an environment, which a learning system has to act in. The goal of the learning system is to find a good action policy to act optimal in regard to the different states of the environment (Russell and Norvig 2010, Chapter 21). RL is inspired by behaviorist psychology and imitates a trial-and-error learning behavior that is observed on humans amongst others, for example when an infant learns how to walk. Infants receive reinforcements for their walking trials, either reward in form of moving forward or punishment in form of falling down. Learning is different than in the supervised case as correct input-output-pairs are not readily available and the learning system has to find a good action policy by trial-and-error interactions with the environment.

For each RL step, an environment state is observed and based on this an action is chosen. Afterwards, the learning system observes a new environment state and receives a reward or punishment for its action in the previous state. In addition, the search for a good policy includes the problem of finding a balance between exploration of unknown regions and exploitation of the current knowledge. If the learning system never explores, then it will be stuck with the current probably sub-optimal policy. On the

other hand, when it explores too much, for example by selecting all its actions randomly, then it will not be able to perform well for the learning task. Furthermore, a taken action influences generally not only the current but also all future reinforcements, which are therefore partially dependent on past actions and called delayed rewards (Sutton and Barto 1998). The overall goal of RL is then to find an action policy that maximizes not the immediate but the long-term reward.

Markov Decision Process

A RL learning problem can be formalized as a solution of a Markov decision process (MDP). An MDP models the decision process in a mathematical framework that is defined by a tuple $\langle S, A, T, R\rangle$ and the environment is assumed to be fully observable (van Otterlo and Wiering 2012). Here, S is the state and A the action space of the environment. $T : S \times A \times S \to [0, 1]$ is a state transition probability function, where $T(s, a, s')$ is the probability to end up in a next state s' when an action a is executed in state s. A reward function R defines what reinforcement is received at each state in the environment. The reward function can be described in several interchangeable ways depending on what is most convenient for the applied learning problem. The possible descriptions include $R : S \to \mathbb{R}$ when reward is given for being in a state s, $R : S \times A \to \mathbb{R}$ when reward is given for choosing an action a in s and $R : S \times A \times S \to \mathbb{R}$ when reward is given for a transition from s to s'. I will mainly consider the reward function defined as $R : S \times A \to \mathbb{R}$.

Online Q-Learning

I then consider online RL, where a learning system has to choose an available action a_t at each time step t based on an action policy $\pi(s_t)$ that is dependent on the current state of the environment s_t. After executing the selected action a_t, the environment changes to the new state s_{t+1} and an immediate reward $r_t = r(s_t, a_t)$ is received. The learning system aims to maximize the accumulated sum of discounted rewards that is defined as

$$R^{\pi}(s, a) = \sum_{t=0}^{\infty} \gamma r_t. \tag{2.2}$$

Here, $\gamma \in [0, 1]$ is a discount factor that regulates the importance of earlier rewards in respect to future rewards. A good prediction of the accumulated sum is provided by the expected value

$$Q^{\pi}(s, a) = E[R^{\pi}(s, a)]. \tag{2.3}$$

The so called Q-function $Q^\pi(s, a)$ is well-known as the action-value function. It describes the expected return that a learning system receives when it executes an action a_t in state s_t and then follows an action policy π (Sutton and Barto 1998).

The Q-function is applied in one of the most effective and popular RL algorithms, Q-learning, where the learning system has to find a good action policy $\pi(s)$ from delayed rewards when transition and reward function are unknown. Q-learning was first proposed by Watkins (1989). Based on the Bellman optimality equation, a learning system approximates an optimal action-value function Q^* by sampling values for a current estimation $Q^\pi(s, a)$ with

$$q(s_t, a_t) = r(s_t, a_t) + \gamma \max_a Q^\pi(s_{t+1}, a). \tag{2.4}$$

Here, $\max_a Q^\pi(s_{t+1}, a)$ is the expected maximal return corresponding to the next state s_{t+1}. Then, a current estimate of the optimal policy can be derived from the approximated Q-function at some learning stage with

$$\pi(s) = \arg\max_a Q^\pi(s, a). \tag{2.5}$$

For simple learning problems, the estimated Q-function values can be represented by a look-up table. But for more complicated problems, it is common to use some function approximator for the representation of the Q-function.

2.2 Artificial Neural Networks

There are different learning systems that can be applied to ML tasks, but this thesis concentrates on the application of artificial neural networks (ANN) in the context of online ML. ANNs are inspired by biological neural networks (especially the brain) imitating their behavior mostly in a simplified manner. ANNs infer the unknown functional relationship of the learning task by processing observed training data. Characteristics of ANNs can be mainly divided into two categories, architecture and learning properties (Jain *et al.* 2014), which have both a big influence on the learning performance. With an appropriate choice of network architecture and learning properties, ANNs are capable of approximating functional relationships even in complex problems quite accurately.

Here, architecture refers to the topology of the network, which consists of many interconnected processing units, often referred to as neurons, arranged in a layered structure. Generally, the network consists of an input and output layer interjected by one or multiple hidden layers. Depending

on the network topology, different layers can be sparsely or densely connected to each other. One of the most popular ANN architectures is the feed-forward ANN, where the processing units are arranged in successive layers with an unidirectional information flow (Saad 1998). The main alternative architecture is the one of recurrent networks, where a bidirectional information flow is established by additional feed-back connections. Yet, I will consider only feed-forward networks in the following. Another architectural property is the number of processing units in each layer. While the number of input and output processing units is equal to the dimensionality of the responding data, the number of hidden units has to be chosen by the user. The number of hidden units is also often referred to as model complexity and plays a crucial role in the learning performance of the ANN.

The learning properties of an ANN refer mainly to the training algorithm that influences how network parameters are updated in order to estimate a good prediction model for the learning task. This also includes whether the network is updated in batch or incrementally. The learning performance of ANNs is depending a lot on the selected training algorithm and its parameters, especially for online learning tasks. For online regression tasks, many ANN training algorithms apply gradient based methods in regard to a differentiable error measure. They can be incrementally updated and have proven to be an efficient technique (Saad 1998).

In truly online learning schemes, data samples are received sequentially over time and are discarded directly after learning. This implies a limited prior knowledge about the task domain and makes tuning of the network and learning parameters much more difficult. It also gives rise to several problems as the proneness to negative interference and the difficult choice of an appropriate model complexity. These problems are explained in more detail in the following subsections 2.2.1 and 2.2.2 respectively. Finally, several network architectures are considered in 2.2.3 that have the potential to deal with the here described challenges in regard to online ML.

2.2.1 Negative Interference

Negative interference is a well known phenomenon that has been observed frequently since the early days of neural network training (McCloskey and Cohen 1989). Often also referred to as catastrophic interference or catastrophic forgetting, negative interference describes the phenomenon of previously learned information being forgotten in favour of new training samples that are processed at a later point of time. Although negative interference is recognized in different learning situations, it is especially a problem in truly online learning schemes. Here, data samples are received over time, processed incrementally and directly discarded after learning. The end objective of training is to estimate a model that reduces the error for

all previously received sample data points. Yet, the ANN updates its parameters to reduce the prediction error for the observed training sample at each time step, which does not necessarily imply that the error for all past samples is also reduced. If the training objective is not achieved due to forgetting effects, negative interference has occurred. Furthermore, the properties of truly online learning schemes make it difficult to ease the effects of negative interference by common approaches, including the use of validation data sets, memorizing all data, retraining as well as the use of prior knowledge about the learning environment (Schaal and Atkeson 1998). In order to achieve good learning performance in these scenarios, it is then important that the ANN itself possesses robust properties against negative interference.

Negative interference can be considered as a natural side-effect of the generalization ability of neural networks (Schaal and Atkeson 1998). Generalization refers to the network's ability to generalize from training data to unseen data samples by providing approximately accurate predictions of the target output for a given input. Since it is infeasible to observe the whole state space for continuous learning problems, generalization ability is an important property of ANNs. It is a kind of positive interference that is achieved by allowing parameters of the network to have non-local effects, or in other words global behavior. Yet, when these non-local effects reduce the overall learning performance more than they can help improve it, negative or even catastrophic interference occurs. Therefore, neural networks with global learning properties are more prone to negative interference than networks with local properties. If learning properties are local, then ANNs are able to keep the negatively interfered areas of the network small.

In environments where negative interference is a problem, a trade-off between global (potential generalization ability) and local (robustness against negative interference) learning behavior is necessary. Completely local presentations, e.g. look-up tables, have no problems with negative interference but also possess no ability to generalize. ANNs that employ local receptive field based structures are a popular alternative. They provide a certain degree of generalization ability within each receptive field but also localize the learning problem providing better robustness against negative interference in comparison with global basis function networks (e.g. multi-layer perceptron with sigmoidal activation functions). Overall, ANNs with local properties are robuster and preferable in online learning settings, where networks with a distributed (global) nature might fail to perform well.

Major causes of negative interference can be found in the training data

distribution, where ANNs are used to approximate a functional relationship between received input-output data pairs. In the standard regression model, it is generally assumed that the input data are distributed according to a probability density $p(x)$ and the input-dependent outputs are distributed according to a conditional probability $p(y|x)$. It was stated in Schaal and Atkeson (1998) that non-stationarity of the learning tasks is a major cause for negative interference and two types can be distinguished: a change in the functional relationship $p(y|x)$ between an input x and output y or a change in the input distribution $p(x)$. Especially the second cause is often reported as the reason for negative interference in online learning schemes. This reasoning about causes can be extended by considering the often made assumption of ANN training algorithms to receive training data samples independently identically distributed (i.i.d.) from a stationary data distribution. A changing input distribution $p(x)$ over time is one example of non-i.i.d. data. In real-world applications, data are often non-i.i.d. and learning performance of ANNs is then affected by negative interference when no preparative measurements are taken (Goodrich 2015; Celaya and Agostini 2015). Other examples of non-i.i.d. data include distributions where data samples are received as a succession of correlated inputs, e.g. in time series, or imbalanced distributions where samples of one region of the input space are observed much more often than in other regions. In case of imbalanced data distributions, the dominant number of input data samples from one region causes the network to forget what has eventually been learned in less frequently visited regions. This is also an issue in reinforcement learning, where some regions of the input space become less visited after learning a good policy. Yet, the network should not forget information about these regions otherwise it will eventually start to visit them again after negative interference has occurred. So, even when ANNs possess a local architecture, it is necessary to further improve the training algorithm when dealing with non-i.i.d. data to avoid negative interference.

In negative interference prone environments, the adaptability of the network model is also an important property of the ANN training algorithm that ensures a successful learning performance. ANN models are often initialized randomly and its parameter updates depend a lot on the model initialization at the first stage of training, where the learning error is still high. Therefore, large updates are necessary for a rapid adaptation to the received information. Yet, large updates lead to quick forgetting of previously received information. So, it becomes necessary to perform adaptation more slowly in later stages of training. Then, old information can be remembered and learning converges. A trade-off between these two learning modes is necessary, which becomes even more difficult to decide on in non-stationary environments. In non-stationary environments, a fast adaptation

of the network model is desirable for newly visited input regions, but on the same time the previously learned information of other regions should be remembered without catastrophic forgetting. In neuroscience, a similar trade-off between adaptation to new information and remembering of old information is known as the stability versus plasticity dilemma (Abraham and Robins 2005). Here, stability refers to the ability of biological systems to store knowledge for a long period of time without forgetting. On the other hand, plasticity refers to the ability of the biological system to learn new information. The knowledge about learning mechanisms in biological systems is rather limited, but generally biological systems do not seem to suffer much from negative interference. For ANNs, it is also desirable to improve them so that they can deal better with negative interference prone learning environments.

2.2.2 Model Complexity Selection

The ANN model complexity is another important factor that has a high influence on the learning performance. It is determined by the number of free parameters in the network architecture, where free parameters are normally found in the hidden layers and connections of the network organized in processing units. The number of processing units is considered optimal when the best possible learning performance is achieved (Bishop 2006, Chapter 1). Yet, the optimal model complexity of the network is usually unknown and approaches for model complexity selection have to be considered.

A major concern is that a non-optimal model complexity will result in either under- or over-fitting, where especially over-fitting is considered a problem. When a network model has too many free parameters, it starts to fit not only the underlying functional relationship but also noise in the training data. Then, the generalization ability of ANNs decreases and the accurate prediction of newly received samples becomes difficult. Over-fitting is also a major reason, why it is bad practice to evaluate the learning performance of an estimated ANN based on the training data, since this does not give an accurate estimation of the generalization performance. On the other hand, if the model complexity is too low, then the model will under-fit and not be able to approximate the underlying relationship of the data. Therefore, a trade-off between over- and under-fitting is necessary, which is also known as the bias-variance dilemma (Geman *et al.* 1992). Assuming that an ANN is trained separately on several data sets, bias refers to how much the ANN's average prediction over all data sets differs from the target function. A high bias (implying a low variance) is then equal to under-fitting, meaning that the ANN's model complexity is too low to represent the learning problem appropriately. Over-fitting is represented by

a high variance and low bias, where the variance measures to what extent the predictions of the separately trained network models vary around the average. A high variance also implies that the estimated network model is sensitive to the applied training data set (Bishop 2006, Chapter 3). Overall, over-fitting is often a bigger concern in the literature about model complexity selection but in general model complexity selection considers the best trade-off between over- and under-fitting.

The model complexity selection problem is a well-known issue when dealing with ANNs, and many methods have been proposed to deal with the problem. The simplest approach is an extensive trial-and-error search over different model complexities, where the model complexity with the best generalization performance is chosen. Yet, this approach is computationally expensive and difficult to apply to online learning scenarios because data sets need to be available in advance. Also, there is the possibility of over-fitting in regard to the used training and test data. Another solution is k-fold cross-validation, where $\frac{k-1}{k}\%$ of the training data are used for the network estimation and the rest is used as a validation set to evaluate the generalization performance. This process is repeated k-times with a changing fold used as validation set. Cross-validation (CV) can mitigate the problem of choosing an over-fitted network model, but also increases the computational expenses by a factor k. Also, k is an open parameter that needs to be selected based on the judgment of the user. When the value of k is too small or too large, high bias or variance becomes a problem and the model selection might not be optimal (Florido *et al.* 2012). Another difficulty in k-fold CV is the appropriate distribution of the data in the folds, especially when the data samples are not i.i.d. Furthermore, various criteria, including Akaike information criterion (AIC) and Bayesian information criterion (BIC), have been proposed for model selection that are theoretically founded in information theory and aim to provide an evaluation for the goodness of fit of different model complexities based only on the training data. These criteria also attempt to penalize more complex models to compensate for the tendencies of over-fitting. They do however not take the uncertainty in the model parameters into account, which leads in practice often to overly simple models (Bishop 2006, Chapter 1). Although all of these methods are commonly considered for ANN model complexity selection, they are not suited for online learning tasks since data samples have to be observed in advance.

Alternatively, heuristic criteria can be applied for model complexity selection in online learning tasks. Since only one data sample is observed at each time step and prior knowledge is limited, it is difficult to decide

on a static model complexity before training. Instead, the network complexity is adapted dynamically during learning based on available information, which consists only of the currently observed data sample and the estimated network model. Mainly, two approaches can be distinguished for dynamic model selection: constructive and pruning approaches. For constructive approaches, the ANN model is initialized with a limited number of processing units that are then increased during learning. Based on a heuristic criteria, it is judged whether increasing the model complexity can bring any major improvement. A new processing unit is added either by producing a new one or by splitting a relevant unit into two. A difficulty with pure constructive approaches is to find an appropriate stopping criteria, because networks tend to increase the number of units steadily with increasing number of processed data. Also, it is not possible to delete units that have been created but became unnecessary at a later point of training, which then leads to unnecessarily high model complexities. For pruning approaches, ANNs are initialized with a large number of units at the beginning and unnecessary units are deleted during the training process. One disadvantage of pruning approaches is that some prior knowledge about the learning task is necessary to ensure the initial number of units is sufficient. Also, the high model complexity at the beginning introduces a high computational burden until the model complexity starts to decrease. A third possibility is a hybrid construction and pruning approach, which aims to combine the strong points of both. All three approaches dynamically adapt the ANN's model complexity during learning. The ANN then does not take a predetermined form but adjusts the number of free network parameters accordingly. Learning with dynamically adaptive models is therefore sometimes referred to as nonparametric regression. These approaches require generally larger sample sizes than parametric models where the model complexity is fixed during learning, because the data must supply enough information for the estimation of both model complexity and model parameters. In online learning tasks, especially the hybrid approach is useful since the model complexity can be dynamically adapted during training according to the needs of the learning tasks.

2.2.3 Networks with a Receptive Field Based Architecture

For online learning tasks, ANNs with local properties have a bigger potential to achieve robust learning performance since the local network structure helps to distribute the effects of negative interference. Therefore, I will concentrate on ANNs with local properties in this thesis, where a special focus is laid on networks with a receptive field based activation of network

units in restricted regions of the input space. Similar localizing properties can also be found in actual biological nervous systems, for example somatosensory systems where regions of the skin behave like receptive fields. Three types of receptive field based networks are considered in more detail in the following.

Radial Basis Function Networks

Radial basis function networks (RBFN) are one popular type of feed-forward neural networks that has been first proposed by Broomhead and Lowe (1988). Its main ideas are based on the mathematical work into the theory of approximating continuous functions by interpolating across known lattice points (Micchelli 1984; Powell 1987). RBFN's popularity is related to the fact that while the network architecture is quite simple, it is possible to apply it to a broad range of learning tasks (Lowe 2015). The RBFN architecture has three layer with one hidden layer sandwiched between the input and output layer. For RBFNs, received inputs are first transformed by non-linear activation functions of the hidden units and then further transformed linearly by some weight vectors to obtain the predicted network output. For an input x, an output y is then given by a weighted linear combination of non-linear basis functions

$$y = \sum_{i=1}^{M} \omega_i \phi(||x - c_i||), \tag{2.6}$$

where M is the number of units in the hidden layer, ω_i is a weight connecting the i-th hidden unit with the output layer and $\phi(||x - c_i||)$ is a radial basis function (RBF) centered at some discrete point c_i. $||\cdot||$ denotes some distance, where the Euclidean norm is the most popularly used (Musavi *et al.* 1992). There are many possible choices for the RBF, including e.g. Gaussian, logistic, linear functions or thin-plate splines (Wu *et al.* 2012). Yet, the most popular is the Gaussian RBF, because it is not only compact and positive but also the only factorisable RBF. They are often termed Gaussian RBFN and have been shown to be able to approximate any continuous function to any degree of accuracy if the model complexity of the hidden layer is sufficient (Park and Sandberg 1991). When RBFs have the property that $\phi(r) \to 0$ as distance $r \to \infty$, they possess local behavior with receptive field like hidden units. This applies for Gaussian RBFs among others, and its broad applicability makes Gaussian RBFNs a popular choice for online learning tasks.

In the most fundamental approach, RBFNs are trained in a two-phase strategy, where in the first phase RBF centers are chosen and in the second phase network weights are adjusted. Additional parameters may have to

be considered dependent on the RBF, for example there is a variance σ^2 for the Gaussian RBF. The appropriate selection of RBF centers is critical for the learning performance of the network (Musavi *et al.* 1992). If training data are available and representative of the learning problem, then it is sufficient to apply a randomly selected subset of the input data as centers. Otherwise, this leads however to an undesirable performance of the RBFN. Clustering can be applied as an alternative selection approach for the centers. Here, the training data are grouped into clusters and the prototypes of each cluster are used as RBF centers. Clustering is one of the most popular approaches for RBF center selection and many supervised and unsupervised clustering algorithms have been applied (Du and Swamy 2006). In case of Gaussian RBFs, an additional variance parameter σ^2 has to be chosen that represents the width of each RBF. Usually, some heuristics are applied to choose accurate widths and one popular option uses the same value for all RBFs, which has already universal function approximation capabilities (Park and Sandberg 1991). After the determination of the RBF parameters, the training of the network weights reduces to a linear optimization problem, which can be solved with methods based on the least squares or gradient-descent approaches (Wu *et al.* 2012). In both cases, the goal is to minimize the mean square error (MSE) of the RBFN in regard to the training data. This parameter selection then provides a fast learning procedure with sufficient accuracy for the RBFN in stationary learning tasks.

RBFN is still a subject of active research and many different training methods and extensions have been proposed over the years to fit different learning problems. The most fundamental training approach is the above described two-phase strategy. Yet, it is also possible to train RBFNs in an one-phase strategy, where supervised learning is employed for the estimation of all RBFN parameters. The simplest approach is gradient-descent (Wu *et al.* 2012), where the network is initialized randomly and then improved gradually during training. Another option is to base the initialization on clustering for RBF centers and least squares for the weights, and then use gradient-descent only to refine the learning results. The gradient-descent method can also be applied incrementally, which makes the application of RBFNs to online learning tasks possible and often increases the learning speed compared to batch updates. Yet, gradient-descent tends to be slow in convergence. There are many other options to train the RBFN, and basically all general purpose unconstrained optimization methods are applicable with almost no modification (Wu *et al.* 2012). One of them is the expectation-maximization (EM) algorithm (Dempster *et al.* 1977), an efficient maximum likelihood-based approach for parameter estimation, whose main idea is to split a complex problem into many separate subproblems with smaller scale. It has been applied for example by Langari *et al.* (1997)

together with another modification, applying linear regression weights instead of constant weights. For regression weights, $\omega_i = a_i' x + b_i$ is an input dependent linear function with an regression parameter vector a_i and a bias b_i. Using regression weights can significantly reduce the number of hidden units and has been applied for example to approximate nonlinear dynamic systems (Langari *et al.* 1997; Rojas *et al.* 2002). A possibility to further generalize the network is the use of arbitrary covariance matrices Σ instead of the simple variance σ^2 for Gaussian RBFs (Wu *et al.* 2012). Yet, extensions of the network architecture also imply an increase in network parameters. Therefore, depending on the learning problem a trade-off between using small networks with many adjustable parameters and using large networks with fewer adjustable parameters might be necessary. The here stated extensions of the network architecture and training approaches have the potential to improve learning performance when used in appropriate learning environments.

Another possible extension is the normalized RBFN, where the individual RBF responses are normalized by the sum of all RBFs. This was first proposed shortly after the traditional RBFN by Moody and Darken (1989). The network architecture is then described by

$$y = \sum_{i=1}^{M} \omega_i \frac{\phi(||x - c_i||)}{\sum_{j=1}^{M} \phi(||x - c_i||)}, \tag{2.7}$$

where predicted output y is now obtained by the normalized weighted RBF sum. This could also be reformulated so that the normalization is performed in the output layer. In a normalized RBFN, the traditional roles of weights and submodels (RBFs) are exchanged. While in the unnormalized model the weights ω_i determine how much each unit's submodel (RBF) contributes to the output, here the normalized RBF becomes the contribution determining weight of a submodel ω_i (Stulp and Sigaud 2015). For Gaussian RBF networks, normalized RBFNs are able to outperform the unnormalized ones in terms of training and generalization errors. They exhibit a more uniform error over the training data domain and less sensitivity to the RBF widths (Bugmann 1998). It was also proven by Benaim (1994) that normalized Gaussian RBFNs are capable of universal function approximation. On the other hand, normalized RBFNs loose some of their local characteristics, and there is even the danger of reactivation of RBFs far away from their actual center under certain conditions (Shorten and Murray-Smith 1996). Yet, it has also been reported that this generally does not negatively affect the learning performance. For normalized RBFNs, similar training approaches can be applied as to the traditional RBFN and they are often considered an interesting alternative.

The model complexity selection problem has been widely considered

for RBFNs with a main focus on constructive and pruning approaches or a combination of both. For constructive approaches, the RBFN gradually increases the number of units either according to a growing criterion or until a stopping criterion is satisfied. One of the most famous approaches is the forward orthogonal least squares (OLS) algorithm (Chen *et al.* 1991), which can also be applied incrementally in its modified extension as recursive OLS algorithm (Yu *et al.* 1997). Other approaches include for example network learning based on the cascade-correlation algorithm, sensitivity analysis, splitting of existing units or heuristics based on error-driven rules (Wu *et al.* 2012). Yet, not all of these approaches are applicable to online learning tasks. On the other hand, pruning approaches start with a large network and then prune unnecessary units to decrease the model complexity. Various pruning methods have been proposed for feed-forward networks, which are applicable to RBFNs. This includes weight-decay techniques, the optimal brain damage (OBD) and optimal brain surgeon (OBS) approaches as well as pruning based on regularization techniques (Wu *et al.* 2012). The third option are combined constructive and pruning approaches. They have been often proposed as extensions of existing constructive approaches to deal with their biggest disadvantages: the unnecessarily large model complexity resulting from unneeded units. In regard to online learning tasks, the most important work for RBFN is likely the Resource Allocating Network (RAN) proposed by Platt (1991). The training of RAN is started with zero units and new units are added every time the current network is not able to present a newly observed data sample well. The network parameters are updated with an incremental least mean square method and the sequential properties enable RAN to model nonstationary processes online. Many extensions have been proposed for RAN and some of the most important ones are RAN with extended Kalman filter parameter updates (RAN-EKF, Kadirkamanathan and Niranjan (1993)) and minimal RAN that applies an additional pruning method to RAN-EKF (MRAN, Lu *et al.* (1997)). The growing and pruning algorithm for RBFNs (GAP-RBF, Huang *et al.* (2004)) and its generalization (GGAP-RBF, Huang *et al.* (2005)) are two other RAN-EKF-based sequential learning algorithms. Here, an additional significance notion has been introduced to evaluate the importance of each unit by its statistical contribution for all observed training data so far. The significance notion was then used to decide on adding new units or pruning older ones that became unnecessary. The (G)GAP-RBF algorithms both outperform the original RAN as well as RAN-EKF and MRAN in terms of learning speed, generalization performance and network complexity. Yet, they require a certain number of input data points making application in truly online learning schemes without prior knowledge difficult. Also, the significance calculation can become computationally heavy

for input dimensions bigger than five (Bortman and Aladjem 2009). The main idea of RAN and all its extension is the self-construction of the network from scratch in an unknown environment, which eases the burden of initialization, and they have been successfully applied to many different application areas.

Mixture of Experts

The mixture of experts (ME) is a widely applied framework that was first introduced by Jacobs *et al.* (1991). The framework's main idea is that often function approximation problems are decomposable into soft partitions of the input space, where every partition is then approximated by a simpler expert. The soft partition is managed through input-dependent gates that put one or several experts in charge of small parts of the input space. The experts can be either regression functions or classifiers making the application to both types of learning tasks possible. The soft partition is based on a principle of divide and conquer where experts compete for being put in charge of certain regions of the input space. The individual experts are then able to specialize on a smaller part of the learning problem, and ME has shown powerful approximation properties by combining the knowledge of these simpler experts together. The modular architecture of MEs increases the locality of the learning behavior and makes it therefore a possible candidate for dealing with online learning tasks. In a broader sense, the ME framework is often compared to decision trees, which also work with localized subspaces, and an extension to hierarchical MEs is straightforward (Jordan and Jacobs 1994). Here, I will concentrate however on single layered ME models since it is enough to explain the basic ideas of the framework and it also builds the foundation for the mainly considered Normalized Gaussian network, which will be introduced in more detail in section 2.3.1.

A big advantage of MEs is its foundation in statistic, where experts and gates are combined by a probabilistic model, enabling MEs to be trained easily with well-known techniques including the expectation-maximization (EM) algorithm as well as variational learning or Markov chain Monte Carlo techniques (Yuksel *et al.* 2012). In the following, I discuss the basic probabilistic model employed by the ME framework. The ME model parameters are denoted by $\theta = \{\theta_g, \theta_e\}$, where θ_g is the set of gate parameters and θ_e is the set of expert parameters. The gates and experts can then be combined by the likelihood of observing an output y for a given input x with

$$P(y|x,\theta) = \sum_{i=1}^{M} P(y,i|x,\theta) = \sum_{i=1}^{M} P(i|x,\theta_g)P(y|i,x,\theta_e), \tag{2.8}$$

where M is the total number of experts and $P(y, i|x, \theta)$ is the likelihood of each expert i to represent the data sample (x, y). The ME model parameters are estimated by maximizing the log-likelihood of the conditional probability with one of the above mentioned training methods.

For the ME regression model, it is assumed that the experts are expressed by a Gaussian model, where $P(y|i, x, \theta_e) = N(y|\hat{y}_i(x), \Gamma_i)$ is the probability density function (pdf) of the i-th expert with a mean $\hat{y}_i(x)$ and covariance Γ_i. The expectation of the likelihood is then used to obtain a predicted output of the ME model for an input x by

$$\hat{y} = \sum_{i=1}^{M} g_i(x, \theta_g)\hat{y}_i(x), \tag{2.9}$$

where $g_i(x, \theta_g)$ is the i-th gating function. The gating functions influence which experts are put in charge of a sample (x, y) and therefore function like a weight. In the original proposal of ME, it is assumed that the experts $\hat{y}_i(x)$ are simple linear functions dependent on input x. When experts are nonlinear, it is not possible to solve the maximization in respect to the expert's parameter analytically and further considerations are necessary to update them (Weigend *et al.* 1995).

In the original proposal by Jacobs *et al.* (1991), the gates are defined as softmax functions. Yet, the nonlinearity of the softmax function makes the maximum likelihood estimation of the gate parameters analytically unsolvable. For example, when the ME model parameters are estimated with the EM algorithm, this leads to an additional inner loop of calculations for each EM iteration. Therefore, Xu *et al.* (1995) proposed an alternative ME model with a different gating function to avoid the inner loops at each iteration and make the gate analytically solvable. The alternative ME uses parametric forms from the exponential family, e.g. Gaussians, and the gating function $g_i(x, \theta_g)$ is then given by

$$g_i(x, \theta_g) = \frac{\alpha_i P(x|i, \theta_g)}{\sum_j \alpha_j P(x|j, \theta_g)}, \ \sum_i \alpha_i = 1, \ \alpha_i \geq 0, \tag{2.10}$$

where $P(x|i, \theta_{g,i})$ are pdfs. For an analytical solution, they further have proposed to work with the joint density $P(y, x|\theta)$ instead of the likelihood $P(y|x, \theta)$. With this analytical solvable maximization in respect to both gate and experts, the alternative ME model is then able to converge faster than the original model (Yuksel *et al.* 2012). Another advantage of the alternative model is the improved locality of the gating function. The softmax gating function divides the input space into overlapping regions by soft hyperplanes. This can lead to difficulties for nontrivial function approximation tasks, because inputs that are not close to any of the hyperplanes activate

many of the gates substantially larger than zero. This makes the parameter estimation much more difficult (Ramamurti and Ghosh 1999). On the other hand, if for example Gaussian densities are applied for the alternative ME model, then the input space is partitioned softly by hyper-ellipsoids which possess a more local behavior with each experts' influence concentrating elliptically around their center. This is then another network architecture with a receptive field based concept.

It is possible to apply ME models to online learning tasks by reformulating training algorithms like the EM algorithm to apply updates incrementally instead of batches. For the alternative ME model, this has been considered separately by Xu (1998) and Ramamurti and Ghosh (1999). The former work has also established a connection of the ME framework to the normalized RBFN, while the later work claims to introduce some growing and pruning methods to the alternative ME model for the first time. In the literature, most of the model selection approaches for ME are however based on the original ME model and ideas are mainly extended from other tree-based algorithms, aiming to find an optimal structure of the tree by growing, pruning, exhaustive search or Bayesian model approaches (Yuksel *et al.* 2012).

Receptive Field Weighted Regression

Receptive field weighted regression (RFWR), first proposed by Schaal and Atkeson (1997), is an incremental variant of the locally weighted regression algorithm (Atkeson and Schaal 1995), another learning method that possesses local properties by combining linear models with weighting functions. RFWR additionally uses some ideas of nonparametric statistics as it is allocating or pruning processing units dynamically as necessary, making its model complexity easily adaptable in online learning tasks. The main difference compared with the previously discussed approaches is that each of the local receptive fields, here presented by Gaussian kernels, is trained in isolation without possessing any knowledge about the other local models in the network. For training, the local experts are updated independently and incrementally by minimizing a locally weighted leave-one-out cross validation error with the recursive least squares method. The isolated training of the experts makes them truly local learning systems and therefore the network is especially robust against negative interference (Nakanishi *et al.* 2005). In addition, the number of updated experts is limited for each time step resulting in fast learning speed.

The properties of its network training make RFWR a popular choice in robot learning and control tasks. On the down side, the learning method requires tuning of several meta-parameters, which are highly data dependent and can easily lead to over-fitting when not sufficiently identified (Meier *et*

al. 2014). This is also a result of the non-cooperative nature of the learning method. RFWR can be interpreted in relation to other learning methods, e.g. as an ME model where the experts are trained in isolation instead of a competitive environment (Schaal and Atkeson 1998). Only when new predictions are made by the trained network model, its outputs are calculated in a similar manner to the ME framework employing normalized weighted sums of the Gaussian kernels as gates.

Summary

All of the discussed network types are based on a common theme by employing receptive field based network architectures. Yet, the receptive fields do not possess always the same role inside the network architecture. While it takes the role of a sub-model for RBFNs, it can also be the weight of the sub-model as for the ME model and RFWR. Although, this requires a shift of interpretation, the structure of the discussed networks is in general the same. Therefore seen from a model-based perspective (Stulp and Sigaud 2015), it is possible to describe all of these network types with the same generic model

$$f(x) = \sum_{i=1}^{M} \phi(x, \theta_i)\omega_i. \tag{2.11}$$

Here, $\phi(x, \theta_i)$ is some arbitrary basis function and $\omega_i = a_i x + b_i$ is a linear model, where in special cases $a_i = 0$ as for example for the original definition of RBFNs in Eq. 2.6. So, the main differences between the network types reduce to the interpretation of sub-models and weights and the chosen learning algorithms that is used to train the network.

A broad range of learning algorithms has been proposed in the literature. Although, I did not discuss them in much detail, they can broadly be divided into local and global training approaches. Local learning algorithms are more robust against negative interference, especially when interference is explicitly considered as for example it is the case for the RFWR network. These discussions show that the robustness of an ANN is influenced not only by the network architecture but by all learning properties and it is necessary to consider them to ensure favorable performance in negative interference prone environments. Interestingly, robustness has never been discussed much in regard to the applied model selection approaches.

2.3 Normalized Gaussian Networks (NGnet)

In this thesis, I lay the main focus on the normalized Gaussian network (NGnet) as proposed by Sato and Ishii (2000), which can be interpreted in

regard to the explanations in 2.2.3 as a normalized RBFN with linear regression weights and training based on the alternative ME framework. In the following, I will explain the network architecture and training method based on the online EM algorithm in detail. In addition, several extensions that have been proposed to improve learning stability of the NGnet are also explained.

2.3.1 Network Architecture

The Normalized Gaussian network (NGnet) transforms an N-dimensional input vector x to a D-dimensional output vector y with

$$y = \sum_{i=1}^{M} N_i(x)\tilde{W}_i\tilde{x}. \tag{2.12}$$

$$N_i(x) \equiv G_i(x)/\sum_{j=1}^{M} G_j(x) \tag{2.13}$$

$$G_i(x) \equiv (2\pi)^{-N/2}|\Sigma_i|^{-1/2}\exp\left[-\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i)\right] \tag{2.14}$$

The model softly partitions the input space into M local units with the normalized Gaussian functions N_i. $\tilde{W}_i \equiv (W_i, b_i)$ is a $D \times (N+1)$-dimensional linear regression weight matrix with $\tilde{x}' \equiv (x', 1)$ where prime $(')$ denotes a transpose. Interpreting this architecture based on the ME framework, $N_i(x)$ is then equal to the i-th gating function of the ME model while $\tilde{W}_i$ is the i-th units expert.

2.3.2 Online EM Algorithm

A stochastic interpretation has been first proposed for an alternative ME framework by Xu *et al.* (1995) that can be used to interpret normalized RBFNs as ME model. The unknown model parameters are then estimated by maximum likelihood estimation based on the log-likelihood of the observed in- and output data (x, y). Here, the Expectation-Maximization (EM) algorithm is applied for parameter estimation. An offline approach has been proposed by Xu *et al.* (1995) that was later adopted to an online EM algorithm by Sato and Ishii (2000).

For the NGnet, a stochastic model is defined by the following probability distribution for a complete event (x, y, i) (Xu *et al.* 1995)

$$P(x, y, i|\theta) = (2\pi)^{-\frac{D+N}{2}} \sigma_i^{-D} |\Sigma_i|^{-\frac{1}{2}} M^{-1} \times \exp\left[-\frac{1}{2}(x - \mu_i)' \Sigma_i^{-1} (x - \mu_i) - \frac{1}{2\sigma_i^2}(y - \tilde{W}_i \tilde{x})^2 \right], \tag{2.15}$$

where $\theta \equiv \{\mu_i, \Sigma_i, \sigma_i^2, \tilde{W}_i | i = 1, ..., M\}$ is the set of model parameters that have to be estimated. The prior probability α_i of the ME framework is represented here by an equal selection probability of each unit $P(i|\theta) = \frac{1}{M}$. For the online EM, the parameters are updated with the following E- and M-step.

E (Estimation) Step:

Given the current estimator $\theta(t-1)$, the posterior probability $P_i(t) \equiv P(i|x(t), y(t), \theta(t-1))$ evaluates how likely the i-th unit generates the current observation $(x(t), y(t))$:

$$P_i(t) \equiv P(i|x(t), y(t), \theta(t-1)) = \frac{P(x(t), y(t), i|\theta(t-1))}{\sum_{j=1}^{M} P(x(t), y(t), j|\theta(t-1))}. \tag{2.16}$$

M (Maximization) Step:

The expected log-likelihood has to be maximized with respect to the model estimation θ, which can be done by updating the model parameters θ at each time step t with

$$\mu_i(t) = \langle\!\langle x \rangle\!\rangle_i(t) / \langle\!\langle 1 \rangle\!\rangle_i(t) \tag{2.17}$$

$$\Sigma_i^{-1}(t) = [\langle\!\langle xx' \rangle\!\rangle_i(t) / \langle\!\langle 1 \rangle\!\rangle_i(t) - \mu_i(t)\mu_i'(t)]^{-1} \tag{2.18}$$

$$\tilde{W}_i(t) = \langle\!\langle y\tilde{x}' \rangle\!\rangle_i(t) [\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t)]^{-1} \tag{2.19}$$

$$\sigma_i^2(t) = \frac{[\langle\!\langle |y|^2 \rangle\!\rangle_i(t) - Tr(\tilde{W}_i(t) \langle\!\langle \tilde{x}y' \rangle\!\rangle_i(t))]}{D \langle\!\langle 1 \rangle\!\rangle_i(t)} \tag{2.20}$$

The parameter updates in Eq. (2.17)-(2.20) include symbols $\langle\!\langle \cdot \rangle\!\rangle_i(t)$ that denote weighted accumulated observations $(x(t), y(t))$ until the current time step t.

Step-wise Updates with Time-Dependent Discount

The online EM is derived from a batch update approach (Xu *et al.* 1995) for that the parameters θ are updated after seeing all observed data samples (x, y), where the total number of data samples is T. For the batch updates, the weighted accumulators were originally formulated as weighted means over all data samples

$$\langle\!\langle f \rangle\!\rangle_i(T) \equiv \frac{1}{T} \sum_{t=1}^{T} f_t P(i|x(t), y(t), \bar{\theta}), \tag{2.21}$$

where $f \equiv f(x, y)$ and $f_t \equiv f(x(t), y(t))$ are used as abbreviations and $\bar{\theta}$ are the estimated model parameters. This has been reformulated for incremental updates to a weighted mean where the estimator is changed after each observation and a current parameter estimation is represented by $\theta(t)$. The weighted mean is then replaced by

$$\langle\!\langle f \rangle\!\rangle_i(T) \equiv \eta(T) \sum_{t=1}^{T} \left(\prod_{s=t+1}^{T} \lambda(s) \right) f_t P(i|x(t), y(t), \theta(t-1)) \tag{2.22}$$

$$\eta(T) \equiv \left(\sum_{t=1}^{T} \left(\prod_{s=t+1}^{T} \lambda(s) \right) \right)^{-1}. \tag{2.23}$$

Additionally, a time-dependent discount factor $\lambda(t)$ has been introduced where $(0 \leq \lambda(t) \leq 1)$. It plays an important role in discarding the effect of old learning results that were employed to an earlier inaccurate estimator. The factor has to be chosen so that $\lambda \to 1$ when $t \to \infty$ to fulfill the Robbins-Monro condition for convergence of stochastic approximations (Kushner and Yin 1997). It has been shown that the discount factor helps to improve convergence speed (Sato 2000). $\eta(T)$ is a normalization coefficient that has a similar function to $\frac{1}{T}$ in the batch update above. A step-wise equation of the weighted mean can then be obtained for every time step t with respect to the posterior probability $P_i(t) \equiv P(i|x(t), y(t), \theta(t-1))$:

$$\langle\!\langle f \rangle\!\rangle_i(t) = (1 - \eta(t)) \langle\!\langle f \rangle\!\rangle_i(t-1) + \eta(t) P_i(t) f_t, \tag{2.24}$$

$$\eta(t) = \left(1 + \frac{\lambda(t)}{\eta(t-1)} \right)^{-1}. \tag{2.25}$$

I will often refer to the discount factor $\lambda(t)$ as *global forgetting* (GF) in the thesis, since all units forget the same amount of information at each time step t with no further consideration of how far they are actually from the

received training data sample $(x(t), y(t))$.

Step-wise Updates with Weight-Dependent Discount

In Celaya and Agostini (2015), it was observed that all the parameter updates in Eq. (2.17)-(2.20) cancel out the normalizations of the weighted means $\langle\!\langle f \rangle\!\rangle_i(t)$. It is possible to eliminate the normalization and accumulate observed information in weighted sums instead of weighted means with

$$\langle\!\langle f \rangle\!\rangle_i(t) = \lambda(t)\langle\!\langle f \rangle\!\rangle_i(t-1) + P_i(t) f_t, \tag{2.26}$$

omitting the normalization coefficient $\eta(t)$. The weighted sums are then only dependent on the discount factor $\lambda(t)$ and can be used as basis to derive a step-wise update equation with a weight-dependent discount factor instead of the time-dependent one. The time-dependent discount factor $\lambda(t)$ can be problematic in applications where data are not i.i.d., because the network model updates only units in the same region as the received data sample while it forgets old learning results over the whole input space. This can lead to problems with negative interference. Therefore, a new weight-dependent discount factor has been derived to localize the forgetting of old learning results in regard to updates with new information (Celaya and Agostini 2015).

The derivation of the weight-dependent discount factor is reviewed shortly in the following. For a weight-dependent discount, learned information should be forgotten only as much as new updates are received. Therefore, the model updates have been based exclusively on weights, and the notation of Eq. (2.26) is changed to use a weight-dependent index with $\omega_k = P_i(t)$ as the k-th received weight:

$$\langle\!\langle f \rangle\!\rangle(\omega_{k-1} + \omega_k) = \Lambda(\omega_k)\langle\!\langle f \rangle\!\rangle(\omega_{k-1}) + \Omega(\omega_k) f_t. \tag{2.27}$$

Here, $\Lambda(\omega_k)$ and $\Omega(\omega_k)$ are a weight-dependent forgetting and update factor respectively. The two factors are unknown functions of ω_k that have been determined based on the following conditions. For a full update, $\Lambda(\omega_k)$ and $\Omega(\omega_k)$ should be $\Lambda(1) = \lambda(t)$ and $\Omega(1) = 1$, reducing to the same values as for a full update of Eq. (2.26). On the other hand, a weighted sum $\langle\!\langle f \rangle\!\rangle$ should remain unaltered when $\omega_k = 0$. Additionally, a consistency condition need to be fulfilled, imposing that a weighted sum $\langle\!\langle f \rangle\!\rangle$ updated once with value f_t and weight $(\omega_k + \omega_{k+1})$ must be the same as $\langle\!\langle f \rangle\!\rangle$ updated twice with f_t and weights ω_k and ω_{k+1}. In the following, the two cases are expressed based on Eq. (2.27):

$$\langle\!\langle f \rangle\!\rangle(\omega_{k-1} + (\omega_k + \omega_{k+1})) = \Lambda(\omega_k + \omega_{k+1})\langle\!\langle f \rangle\!\rangle(\omega_{k-1}) + \Omega(\omega_k + \omega_{k+1}) f_t$$

(2.28)

$$\begin{aligned}\langle\!\langle f \rangle\!\rangle((\omega_{k-1}+\omega_k)+\omega_{k+1}) &= \Lambda(\omega_{k+1})\langle\!\langle f \rangle\!\rangle(\omega_{k-1}+\omega_k) + \Omega(\omega_{k+1})f_t \\ &= \Lambda(\omega_{k+1})\Lambda(\omega_k)\langle\!\langle f \rangle\!\rangle(\omega_{k-1}) + (\Lambda(\omega_{k+1})\Omega(\omega_k) + \Omega(\omega_{k+1}))\, f_t.\end{aligned} \tag{2.29}$$

Based on Eq. (2.28) and (2.29), the following functional equations were obtained

$$\Lambda(\omega_k + \omega_{k+1}) = \Lambda(\omega_{k+1})\Lambda(\omega_k), \tag{2.30}$$

$$\Omega(\omega_k + \omega_{k+1}) = \Lambda(\omega_{k+1})\Omega(\omega_k) + \Omega(\omega_{k+1}). \tag{2.31}$$

With the additional full update condition, it is possible to obtain a weight-dependent forgetting and update factor from these functional equations, where $\Lambda(\omega_k) = \lambda(t)^{\omega_k}$ and $\Omega(\omega_k) = \frac{1-\lambda(t)^{\omega_k}}{1-\lambda(t)}$. Finally, the notation has been returned to a time based index, and ω_k was replaced with $P_i(t)$. The weight-dependent stepwise update becomes

$$\langle\!\langle f \rangle\!\rangle_i(t) = \lambda(t)^{P_i(t)}\langle\!\langle f \rangle\!\rangle_i(t-1) + \left(\frac{1-\lambda(t)^{P_i(t)}}{1-\lambda(t)}\right) f_t. \tag{2.32}$$

In this thesis, I will often refer to the weighted discount factor $\lambda(t)^{P_i(t)}$ as *localized forgetting* (LF), because the additional weight $P_i(t)$ prevents units far from a current data sample $(x(t), y(t))$ to forget when no new information is received.

2.3.3 Extension for NGnet's Online EM

A few extensions have been proposed by Sato and Ishii (2000) for the NGnet's online EM-algorithm to deal with problems that can occur during learning. One problem is the computational heaviness of the necessary inverse matrix calculations and another problem is that the NGnet's input covariances can become singular. For both problems, some solutions have been proposed which I will also employ in this thesis and explain in the following. Yet, because a different update approach with weighted sums is considered here instead of the weighted means in Sato and Ishii (2000), I will reformulate the necessary equations so that they generalize to both update approaches that have been explained in the last subsection. Therefore, I use a general notation where the forgetting factor is denoted $\Lambda_i(t)$ and the update factor is $\Omega_i(t)$. In the time-dependent discount case, the forgetting factor is either $\Lambda_i(t) = 1 - \eta(t)$ when normalization is applied or else $\Lambda_i(t) = \lambda(t)$,

and the update factor is $\Omega_i(t) = P_i(t)$. For the weight-dependent discount factor, the forgetting factor becomes $\Lambda_i(t) = \lambda(t)^{P_i(t)}$ and the update factor is $\Omega_i(t) = \frac{1-\lambda(t)^{P_i(t)}}{1-\lambda(t)}$. Both extensions are then explained below with the generalized notation.

Avoiding Inverse Matrix Calculation

The online parameter updates in Eq. (2.18) for Σ_i^{-1} and Eq. (2.19) for $\tilde{W}_i$ include inverse matrix calculations at every time step t. This can become computationally heavy with increasing input dimension, but can be avoided by employing a recursive formula derived from standard methods. An additional weighted inverse covariance matrix of $\tilde{x}$ is defined as $\tilde{\Psi}_i(t) \equiv (\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t))^{-1}$, where $\tilde{\Psi}_i(t)$ can be recursively obtained by the following step-wise equation:

$$\tilde{\Psi}_i(t) = \frac{1}{\Lambda_i(t)} \left[\tilde{\Psi}_i(t-1) - \frac{\Omega_i(t)\tilde{\Psi}_i(t-1)\tilde{x}\tilde{x}'\tilde{\Psi}_i(t-1)}{\Lambda_i(t) + \Omega_i(t)\tilde{x}'\tilde{\Psi}_i(t-1)\tilde{x}} \right]. \tag{2.33}$$

Then, $\Sigma_i^{-1}(t)$ can be obtained from the following relation with $\tilde{\Psi}_i(t)$.

$$\tilde{\Psi}_i(t)\langle\!\langle 1 \rangle\!\rangle_i(t) = \begin{pmatrix} \Sigma_i^{-1}(t) & -\Sigma_i^{-1}(t)\mu_i(t) \\ -\mu_i'(t)\Sigma_i^{-1}(t) & 1 + \mu_i'(t)\Sigma_i^{-1}(t)\mu_i(t) \end{pmatrix}. \tag{2.34}$$

Also, the linear regression matrix $\tilde{W}_i(t)$ can be calculated with $\tilde{\Psi}_i(t)$ by

$$\tilde{W}_i(t) = \langle\!\langle y\tilde{x}' \rangle\!\rangle_i(t)\tilde{\Psi}_i(t) = \tilde{W}_i(t-1) + \Omega_i(t)(y(t) - \tilde{W}_i(t-1)\tilde{x})\tilde{x}'\tilde{\Psi}_i(t). \tag{2.35}$$

Regularization of Covariance Matrix

It has been assumed for the derivation of the parameter updates with the EM-algorithm that the input covariance matrix is always a positive semi-definite matrix and does not become singular so that its matrix inverse does exits. Yet, this can not always be ensured when the learning algorithm is applied to a learning tasks, especially real testbeds. In Sato and Ishii (2000), a regularization method has been proposed for the covariance matrix to deal with this problem. It has been derived based on the offline updates and then applied to the online EM-algorithm, but for keeping the explanation short only the necessary changes for the online EM-algorithm are explained.

For the online EM-algorithm, the regularization can be performed with the following changes. A regularized Σ_i^{-1} is obtained from the relation (2.34) by using a regularized $\tilde{\Psi}_i$.

$$\tilde{\Psi}_i(t) = \left(\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t) + \alpha \langle\!\langle \Delta_i^2 \rangle\!\rangle_i(t) \tilde{I}_N \right)^{-1}, \tag{2.36}$$

where $\tilde{I}_N$ is an $((N+1)\times(N+1))$-dimensional matrix defined by

$$\tilde{I}_N \equiv \begin{pmatrix} I_N & 0 \\ 0 & 0 \end{pmatrix} = \sum_{n=1}^{N} \tilde{e}_n \tilde{e}_n'. \tag{2.37}$$

Here, $\tilde{e}_n$ is an $(N+1)$-dimensional unit vector with its n-th element equal to 1 and all other elements equal to 0. A data variance $\Delta_i^2(t)$ can be obtained for the current time step t by

$$\Omega_i(t)\Delta_i^2(t) = \Omega_i(t)|x(t)-\mu_i(t)|^2/N + \Lambda_i(t)|\mu_i(t)-\mu_i(t-1)|^2 \langle\langle 1 \rangle\rangle_i (t-1)/N. \tag{2.38}$$

Using data variance Δ_i^2 and a small constant α, I can obtain the virtual data $\tilde{\upsilon}_n(t)|\{n=1,\ldots,N\}$ with

$$\tilde{\upsilon}_n(t) \equiv \sqrt{\alpha}\Delta_i(t)\tilde{e}_n, \tag{2.39}$$

which is then used to regularize the weighted covariance matrix $\tilde{\Psi}_i(t)$ after the update with Eq. (2.33) by

$$\tilde{\Psi}_i(t) := \tilde{\Psi}_i(t) - \frac{\Omega_i(t)\tilde{\Psi}_i(t)\tilde{\upsilon}_n(t)\tilde{\upsilon}_n'(t)\tilde{\Psi}_i(t)}{1+\Omega_i(t)\tilde{\upsilon}_n'(t)\tilde{\Psi}_i(t)\tilde{\upsilon}_n(t)}. \tag{2.40}$$

After regularization, the linear regression matrix $\tilde{W}_i$ is obtained by using Eq. (2.35).

2.3.4 Dynamic Model Selection for the NGnet

Some unit manipulation mechanisms have been introduced for the time-dependent update approach (Sato and Ishii 2000) that enable the NGnet to dynamically add or prune units during learning and therefore adapt to non-stationary environments. The mechanisms include two types of unit adding, produce and split, and one type of pruning as introduced in the following. When adding new units by production or splitting, it is necessary to initialize these units. In this thesis, I propose some changes to the manipulation decisions, which is explained in chapter 3, but use the new unit initializations as explained below.

Produce

When the current network model cannot accurately represent a newly received data sample $(x(t), y(t))$, then a new unit should be created with the initialization stated below.

$$\mu_{M+1} = x(t) \tag{2.41}$$

$$\Sigma_{M+1}^{-1} = \chi_{M+1}^{-2} I_N, \quad \chi_{M+1}^2 = \beta_1 min_{i=1}^M |x(t) - \mu_i|^2/N \tag{2.42}$$

$$\sigma_{M+1}^2 = \beta_2 max_{i=1}^M \sigma_i^2 \tag{2.43}$$

$$\tilde{W}_{M+1} \equiv (W_{M+1}, b_{M+1}) = (0, y(t)), \tag{2.44}$$

where β_1 and β_2 are chosen to be appropriate positive constants. I will use $\beta_1 = \beta_2 = 1.0$ for my approach.

Delete

When a unit is rarely used, it should be deleted. An indicator for this is the weighted mean of one $\ll 1 \gg_i (t)$ that accumulates information about how much the i-th unit has been used to represent data samples until the current time step t.

Split

The output variance $\sigma_i^2(t)$ is representing the accumulated squared error between the i-th unit's predictions and the real outputs. High variance values are often related to the unit being in charge of a too large partition of the input space and not being able to represent the data accurately. When the output variance is bigger than a split threshold, the unit should be split into two to reduce the size of the partition in charge.

$$\mu_{M+1}(new), \mu_i(new) = \mu_i(old) \pm \beta_3 \sqrt{\xi_1} \psi_1 \tag{2.45}$$

$$\Sigma_{M+1}^{-1}(new), \Sigma_i^{-1}(new) = 4\xi_1^{-1} \psi_1 \psi_1' + \sum_{n=2}^{N} \xi_n^{-1} \psi_n \psi_n' \tag{2.46}$$

$$\sigma_{M+1}^2(new), \sigma_i^2(new) = \sigma_i^2(old)/2 \tag{2.47}$$

$$\tilde{W}_{M+1}(new), \tilde{W}_i(new) = \tilde{W}_i(old), \tag{2.48}$$

where ξ_n and ψ_n denote the n-th eigenvalue and eigenvector of the covariance matrix $\Sigma_i(old)$ with $\xi_1 = \xi_{max}$. β_3 is an appropriate positive constant that regulates the overlap between the split units and is set to $\beta_3 = 0.5$ for my approach.

Chapter 3

Proposed Method

In this chapter, I propose several improvements for the NGnet to achieve better robustness and applicability to online learning tasks (Backhus *et al.* 2017). In Section 2.3.2, I have discussed a learning algorithm with localized forgetting, which has been recently proposed by Celaya and Agostini (2015) and provides an improved robustness of the NGnet's update method in regard to negative interference. This makes it an interesting alternative to a more established update method with global forgetting, which has been shortly introduced in Section 2.3.2. Yet, further improvements are possible for the localized forgetting update method to achieve better robustness and applicability. When network parameters are estimated with the previous localized forgetting update method (Eq. (2.32)), then the discount factor $\lambda(t)$ is not applicable over its full numerical range. Therefore, I rediscuss the derivation of localized forgetting updates in Section 3.1. Furthermore, localized forgetting offers the possibility to reduce the computational complexity of updates, since many units become computationally irrelevant at each time step. I discuss this matter more deeply in Section 3.2. Finally, it is difficult to apply localized forgetting to online learning tasks, where domain knowledge is limited or the environment is non-stationary, since previously only static model selection has been considered. Therefore, I discuss the application of dynamic model selection and additional improvements for it in regard to negative interference in Section 3.3. This also includes the proposal of a new merge mechanism to deal with model redundancies.

3.1 Re-Derivation of Localized Forgetting

In the following, I re-discuss the derivation of an update approach with localized forgetting (Backhus *et al.* 2016a), where localized forgetting has been proposed earlier by Celaya and Agostini (2015) and was already shortly discussed in Section 2.3.2. This discussion is motivated by the dependency of an update factor $\Omega(P_i(t))$ on the discount factor $\lambda(t)$ in Eq. (2.32) with $\Omega(P_i(t)) = \frac{1-\lambda(t)^{P_i(t)}}{1-\lambda(t)}$. I believe this to be bad practice, because the discount $\lambda(t)$ controls forgetting of previously learned information and should not

influence the update weight of new information. Originally, the discount factor has been introduced to speed up convergence by partially forgetting older inaccurate estimations of a network parameter (Sato 2000). The discount factor takes values in the range $0 \leq \lambda(t) \leq 1$. When the localized forgetting update method employs a discount $\lambda(t) = 1$, the update factor becomes however $\frac{1-1^{P_i(t)}}{1-1} = \frac{0}{0}$ and no new information can be learned. Therefore, I derive the update approach with localized forgetting anew from the global forgetting approach in Eq. (2.26). While the previous derivation, explained in Section 2.3.2, has emphasized only weight dependencies, I consider a derivation with dependencies on weight and time.

The new derivation follows the same flow as the previous one, but I keep a time index notation over the whole derivation to emphasize the additional time dependency. The stepwise update in Eq. (2.26) is then rewritten to

$$\langle\!\langle f \rangle\!\rangle_i(t) = \Lambda(P_i(t))\langle\!\langle f \rangle\!\rangle_i(t-1) + \Omega(P_i(t))f_t, \tag{3.1}$$

where a new forgetting factor $\Lambda(P_i(t))$ and update factor $\Omega(P_i(t))$ have to be determined by considering a full update and consistency condition.

I consider a full update condition, where $\Lambda(1) = \lambda(t)$ and $\Omega(1) = 1$, and implying a dependency on time t and the competitive sharing of a data sample by several units. The units divide the responsibility for each data sample $(x(t), y(t))$ between each other according to their representation ability of $(x(t), y(t))$. A full update can therefore only happen at a same time step t for a weighted sum $\langle\!\langle f \rangle\!\rangle_M$ that accumulates information over all M units, or when only one unit represents the data sample to 100% which is rarely happening. If the batch update equation (2.22) is rewritten to an update without normalization and accumulated over all M units, then I get

$$\langle\!\langle f \rangle\!\rangle_M(T) = \sum_{t=1}^{T} \left(\prod_{s=t+1}^{T} \lambda(s) \right) \sum_{i=1}^{M} P_i(t) f_t = \sum_{t=1}^{T} \left(\prod_{s=t+1}^{T} \lambda(s) \right) f_t \tag{3.2}$$

as the full update for global forgetting. Considering the same for local forgetting, I would get

$$\langle\!\langle f \rangle\!\rangle_M(T) = \sum_{t=1}^{T} \left(\prod_{s=t+1}^{T} \left(\prod_{j=1}^{M} \Lambda\left(P_j(s)\right) \right) \right) \sum_{i=1}^{M} \Omega(P_i(t)) f_t = \sum_{t=1}^{T} \left(\prod_{s=t+1}^{T} \lambda(s) \right) f_t. \tag{3.3}$$

Here, global and local forgetting reduce to the same full update as the right terms of the equation show.

Additionally, I have to consider a consistency condition for a shared

weighted sum $\langle\!\langle f \rangle\!\rangle_{i+j}(t-1)$ by unit i and j. Here, an update with value f_t should be the same when updated once with weight $(P_i(t)+P_j(t))$ or twice with weights $P_i(t)$ and $P_j(t)$ at one time step t. For a single update with $(P_i(t)+P_j(t))$, I get

$$\langle\!\langle f \rangle\!\rangle_{i+j}(t) = \Lambda\left(P_i(t)+P_j(t)\right)\langle\!\langle f \rangle\!\rangle_{i+j}(t-1) + \Omega\left(P_i(t)+P_j(t)\right) f_t. \quad (3.4)$$

On the other hand, if $\langle\!\langle f \rangle\!\rangle_{i+j}(t-1)$ is updated twice with $P_i(t)$ and $P_j(t)$, then I get

$$\langle\!\langle f \rangle\!\rangle_{i+j}(t) = \Lambda(P_i(t))\langle\!\langle f \rangle\!\rangle_{i+j}(t-1) + \Omega(P_i(t)) f_t \quad (3.5)$$

$$\langle\!\langle f \rangle\!\rangle_{i+j}(t+1) = \Lambda(P_j(t))\langle\!\langle f \rangle\!\rangle_{i+j}(t) + \Omega(P_j(t)) f_t. \quad (3.6)$$

By inserting Eq. (3.5) into Eq. (3.6), I get

$$\begin{aligned}\langle\!\langle f \rangle\!\rangle_{i+j}(t+1) = \Lambda(P_j(t))\Lambda(P_i(t))\langle\!\langle f \rangle\!\rangle_{i+j}(t-1) \\ + \left(\Lambda(P_j(t))\Omega(P_i(t)) + \Omega(P_j(t))\right) f_t, \quad (3.7)\end{aligned}$$

which is similar to Eq. (2.29) but uses a time index notation instead. This time index notation helps to observe that Eq. (3.7) results in an update that is one step ahead of time and therefore also ignores that units actually divide the update weights for each training data sample $(x(t), y(t))$ at one time step t. Furthermore, the update $\Omega(P_i(t)) f_t$ is already partially forgotten by $\Lambda(P_j(t))$. I want however update twice on the same time step t, so I have to prevent forgetting of partial updates which reduces Eq. (3.7) to

$$\langle\!\langle f \rangle\!\rangle_{i+j}(t) = \Lambda(P_j(t))\Lambda(P_i(t))\langle\!\langle f \rangle\!\rangle_{i+j}(t-1) + \left(\Omega(P_i(t)) + \Omega(P_j(t))\right) f_t. \quad (3.8)$$

The same equation can also be derived from the batch update in Eq. (3.3), when it is rewritten to a stepwise update equation and considered as a weighted sum shared by two units i and j instead of M units.

Finally, I obtain the following two functional equations based on Eq. (3.4) and Eq. (3.8):

$$\Lambda(P_i(t)+P_j(t)) = \Lambda(P_j(t))\Lambda(P_i(t)) \quad (3.9)$$

$$\Omega(P_i(t)+P_j(t)) = \Omega(P_i(t)) + \Omega(P_j(t)). \quad (3.10)$$

For the functional relationship of the forgetting factor in Eq. (3.9), the solution is well known to be of the form $\Lambda(P_i(t)) = c^{P_i(t)}$, where c can be determined by using $\Lambda(1) = \lambda(t)$. The same forgetting factor is derived as for the previous localized forgetting approach with $\Lambda(P_i(t)) = \lambda(t)^{P_i(t)}$. Yet, I derive a new update factor from Eq. (3.10) with $\Omega(P_i(t)) = P_i(t)$, which is actually the same as for the global forgetting approach. The newly derived localized forgetting update approach then complies with all considered dependencies and can add new information to units independently from $\lambda(t)$. The new stepwise update equation for local forgetting becomes

$$\langle\!\langle f \rangle\!\rangle_i(t) = \lambda(t)^{P_i(t)} \langle\!\langle f \rangle\!\rangle_i(t-1) + P_i(t) f_t. \tag{3.11}$$

3.2 Update Precision

The local forgetting update method possesses some properties, which the global forgetting method does not, that make a reduction of the computational complexity possible. In a training approach with global forgetting as proposed by Sato and Ishii (2000), all units have to be updated at every time step because even when a unit's update with new information almost equals zero and therefore could be omitted, each unit definitely forgets a fixed amount of previously learned information. This then results in a change in the unit's parameters. On the other hand, the local forgetting approach sets the forgetting of old information in relation to the actual learned new information. This implies that a unit with a very small update weight at a time step also applies a very small discount, where $\lambda(t)^{P_i(t)} \to 1$ for updates weights $P_i(t) \to 0$. While mathematically seen updates can become very small, the computational precision is limited. When update weights are so small that they are outside of these computational precision limits, the discount factor will equal one and the update weights zero so that the units parameter are the same before and after the update. This raises then the possibility to reduce the computational complexity of the parameter updates by ignoring units whose updates are computationally irrelevant.

Updates with localized forgetting provide a natural approach to reduce the computational complexity for the NGnet. The number of updated units can be reduced according to the precision limits of the implementation environment, which is approximately $1.0E-17$ in this case. Yet, even when update weights are slightly bigger than $1.0E-17$, the influence of these updates is very low for the overall estimation of the unit's parameters and also for the learning performance. In addition, the applied forgetting factor eliminates changes made for very small parameter updates easily with the next larger update. Therefore, it would be interesting to discuss how much

the actual update precision can be reduced without resulting in any larger change of the NGnet's learning performance. If it is possible to reduce the update precision further, then this would result in additional ease of computational complexity while learning performance stays approximately the same. Because it is difficult to decide on an appropriate update precision based on a theoretical discussion only, I will further discuss this matter for experimental results presented in Section 4.2.1 and 4.2.2 of Chapter 4.

Aside from computational complexity, I can also consider the update precision reduction from the viewpoint of a global-local trade-off. The reduced number of updated units implies an increased locality of the update since units with higher update weights are likely to be nearer to the currently observed training data sample. But an increasing locality also implies a decreasing generalization ability, which can affect the learning performance negatively in noisy environments. Local models tend to over-fit more easily since the range of received information is reduced for each unit. This effect has also been noted in the context of the ME framework by Jacobs (1997). ME models softly partition the input space, which was then stated to be an advantage in regard to noise robustness compared with other learning systems that apply a hard partition of the input space. So, when the update precision is cut down too much, the NGnet's learning performance is likely to decrease. Yet, this probably only plays a role for quite aggressive cuts in the update precision since smaller updates' influence is generally low. This additional viewpoint gives further insights into how to interpret the experimental results discussed in Section 4.2.1 and 4.2.2 and based on it how to decide on an appropriate update precision.

3.3 Dynamic Model Selection

Dynamic model selection is an important feature when applying ANNs to online learning tasks, since it is difficult to choose an appropriate static network model complexity and initialization of model parameters without incorporating domain knowledge. Also, learning performance can be highly dependent on both factors. An accurate model selection in advance is normally only possible with extensive trial-and-error studies conducted by the user. Therefore, the application of dynamic model selection can ease the use of NGnets for learning tasks, where finding a good model by hand is difficult. While dynamic model selection has not been considered for an NGnet with localized forgetting, it was previously introduced to an NGnet with global forgetting. In Section 3.3.1, I discuss therefore the adaptation of some unit manipulation mechanisms that have been shortly introduced in Section 2.3.4 and were previously applied to an NGnet with global forgetting. Furthermore, I aim to improve the robustness of the mechanisms in

regard to negative interference and introduce an additional merge mechanism to reduce redundancies in the model.

Dynamic model selection is a popular model selection approach and has been considered for many ANNs with a receptive field based network architecture. It is normally achieved by employing some methods to increase or reduce model complexity by adding or pruning units. In addition, it is possible to apply these methods self-constructively so that the model is build from scratch during learning and initialized with zero units before training. This has the advantage that the initialization problem can be almost completely avoided. I discuss an self-constructing algorithm for the NGnet in Section 3.3.2.

3.3.1 Unit Manipulation Mechanisms

Some unit manipulation mechanisms have been introduced for the NGnet with global forgetting (Sato and Ishii 2000) and are adapted here to apply dynamic model selection to the localized forgetting approach. The adapted unit manipulation mechanisms include a produce, delete and split mechanism, which were discussed shortly in Section 2.3.4. Additionally, I propose a new merge mechanism to reduce redundancy of units and further improve model compactness.

Produce

The probability distribution $P(x(t), y(t)|\theta(t-1)) = \sum_{i=1}^{M} P(x(t), y(t), i|\theta(t-1))$ indicates how accurate the current model parameters $\theta(t-1)$ can represent the newly observed data sample $(x(t), y(t))$. Then, this information can be used to decide whether a new unit should be added or not. The above stated probability distribution includes however the prior probability $P(i|\theta(t-1)) = \frac{1}{M}$, which implies a tendency to add new units more easily with increasing model complexity M. To avoid this, I can omit the prior probability and use only the input output probability distribution $\sum_{i=1}^{M} P(x(t), y(t)|i, \theta(t-1))$ of the NGnet as an indicator for unit construction instead. When the indicator is smaller than a certain threshold $T_{Produce}$, the current network model is not able to represent the currently observed data sample well enough and a new unit is created according to Eq. (2.41)-(2.44).

While the initialization of the new unit's parameters has been discussed by Sato and Ishii (2000), any hints about the initialization of the weighted sums were omitted, which is therefore shortly discussed in the following. When a new unit is created at a time step t, this unit is the only responsible unit for the current sample $(x(t), y(t))$. This can be expressed by an update weight $P_{M+1}(t) = 1$, which then is used to initialize the weighted

mean of one to $\langle\!\langle 1 \rangle\!\rangle_{M+1}(t) = P_{M+1}(t) = 1$. An initialization of $\langle\!\langle 1 \rangle\!\rangle_{M+1}(t)$ is then enough to calculate the initial values of all other weighted sums with the available information according to the functional relationship between weighted sums and unit parameters in Eq. (2.17)-(2.20).

Delete

The importance of an unit is indicated by the weighted sum of one $\langle\!\langle 1 \rangle\!\rangle_i(t)$ that accumulates information about how much the i-th unit has been in charge of the observed data until the current time step t. When the delete mechanism has been first proposed (Sato and Ishii 2000), $\langle\!\langle 1 \rangle\!\rangle_i(t)$ was assumed to be a weighted mean scaled between zero and one by a normalizer $\eta(t) = (1 + \lambda(t)/\eta(t-1))^{-1}$. The normalizer $\eta(t)$ replaced a normalizer $1/T$, which is used in the batch EM-algorithm and T is the total number of samples. T is however unknown in online learning tasks and instead $\eta(t)$ represents an normalizer based on the current data count $1/t$ discounted by $\lambda(t)$.

For the NGnet with localized forgetting, $\langle\!\langle 1 \rangle\!\rangle_i(t)$ is an unscaled weighted sum and cannot be used directly as a reference for the delete mechanism. Therefore, a normalizer has to be introduced before using $\langle\!\langle 1 \rangle\!\rangle_i(t)$ as an indicator and there are several possible candidates for the normalization. The first candidate is the sum $\sum_{i=1}^{M} \langle\!\langle 1 \rangle\!\rangle_i(t)$, which represents the current total number of data samples represented by the trained NGnet. It is also an approximation of t with consideration of the received discounts, and in case of $\lambda(t) = 1$ it would reduce to the same value as $\eta(t)$. While this normalizer evokes a similar effect as the normalizer $\eta(t)$ in Eq. (2.24), it is unlikely to work well in environments where the model complexity is dynamic. All units are initialized with the weighted sum $\langle\!\langle 1 \rangle\!\rangle_i(t) = 1$, which represents their presentation probability of the data sample that they have been added for. The point of time t is not considered for the initialization of newly produced units. With a normalizer as $\sum_{i=1}^{M} \langle\!\langle 1 \rangle\!\rangle_i(t)$, later added units are likelier deleted again soon after their production with increasing t, because the normalization of $\langle\!\langle 1 \rangle\!\rangle_i(t)$ with a large value of the approximated t results in very small values. A second possible normalizer is a separate life time counter for each unit. Then, the unit is only influenced by its own life time and not by the current training time step t. This is however still a global normalization that has similar tendencies as the global forgetting update method. Yet, I want a normalization that performs well even for negative interference prone environments and therefore localize the delete decision further to achieve better robustness.

A third possible normalization candidate is introduced that only increments each units independent time counter when the unit is assumed relevant for a data sample $(x(t), y(t))$. This normalization then provides a localization of the delete mechanism, which helps to improve robustness against negative interference. The time counter of a unit is incremented either when the unit belongs to the updated units at time step t or when it is not updated but the input posterior probability $P(i|x(t), \theta(t)) = \frac{P(x(t)|i,\theta(t))}{\sum_{j=1}^{M} P(x(t)|j,\theta(t))}$ is high enough for the unit to be considered locally relevant. To take each unit's received discount into account, I define the time counter $c_i^{(time)}$ similarly to $\eta(t)$ with

$$c_i^{(time)} = 1 + \lambda(t)^{P_i(t)} c_i^{(time)}. \tag{3.12}$$

But the major difference to $\eta(t)$ is that each unit has its own counter, which is managed independently by the unit and not incremented at each time step. A unit i is then deleted if $\langle\langle 1 \rangle\rangle_i(t)/c_i^{(time)} < T_{Delete}$, where T_{Delete} is a delete threshold. It is necessary to include the discount information into the normalization, since the applied discount evokes that the weighted sums stop growing after reaching a certain limit where applied discount and newly added information become approximately the same for each time step. This is especially a problem when the discount factor converges only slowly to one or does not converge at all. Not including the discount information would then lead to the deletion of highly active units because the time counter is still growing linearly.

Split

The split mechanism is another approach to increase model complexity, which considers units that are in charge of a large partition of the input space but not able to represent data samples well. The output variance of a unit i, $\sigma_i^2(t)$, is an indicator for this as it accumulates information about the squared error between the unit's predictions and the real outputs. High variance values are then the result of a high error accumulation and splitting such units can improve learning performance. In Section 2.3.4, a split mechanism has been discussed that employs a static threshold parameter to decide whether a unit is split or not. For a static threshold, it is however difficult to find an appropriate setting in non-i.i.d. learning tasks, especially when the data distribution is imbalanced and different regions are in need of different threshold parameters.

For my split decision, I compare each unit's $\sigma_i^2(t)$ with a dynamic threshold parameter that is calculated from local information and therefore attempts to localize the split mechanism. Here, the threshold parameter is calculated by considering only output variances of nearest neighbors of the

split candidate. When a unit's output variance is considerably bigger than the biggest variance of its neighbors, the unit is split and the unit parameters are initialized according to Eq. (2.45)-(2.48). Additionally, I have to decide on an initialization for $\langle\!\langle 1 \rangle\!\rangle_{new}$ so that the rest of the weighted sums can be initialized accurately based on the available information. Because the sum $\sum_{i=1}^{M} \langle\!\langle 1 \rangle\!\rangle_i(t)$ over all units M is similar to a discounted counter of the current total number of time steps T, I initialize $\langle\!\langle 1 \rangle\!\rangle_{new}(t) = \langle\!\langle 1 \rangle\!\rangle_{old}(t)/2$ for the two split units so that the sum $\sum_{i=1}^{M} \langle\!\langle 1 \rangle\!\rangle_i(t)$ remains the same.

The algorithm below describes the flow of the dynamic threshold calculation for my split mechanism.

1: Number of nearest neighbors: $NoNN = M/Div_{(NN)}$
2: **for all** candidate units i **do**
3: Find the $NoNN$ nearest neighbors of unit i
4: ${\sigma^2_{NN}}^{(max)} = \max\{\sigma^2_j, j = 1, 2, \cdots, NoNN\}$
5: Calculate dynamic split threshold:
$T_{Split} = {\sigma^2_{NN}}^{(max)} \cdot C$
6: **if** $\sigma^2_i > T_{Split}$ **then**
7: Add unit to splitting candidates vector V_{Split}
8: **end if**
9: **end for**
10: Split all candidates in V_{Split}

Here, $Div_{(NN)}$ and C are two variables that have to be set by the user. $Div_{(NN)}$ is regulating the number of nearest neighbors involved in the splitting decision in relation to the current model complexity M of the NGnet. $Div_{(NN)}$ is set to 6 for my method. On the other hand, C is a constant that regulates when a unit is considered too error-prone compared with its neighbors and has to be split. C is set individually according to the needs of the experiments and information about the chosen values can be found in Chapter 4. Nearest neighbor search is based on the distance between two units. The $NoNN$ units with the shortest distances are considered as nearest neighbors of unit i and the Euclidean distance is used for distance calculation. Because nearest neighbor search is computationally expensive, I limit the number of considered split candidates to actually updated units at each time step.

Merge

Additionally, I propose a new merge manipulation mechanism (Backhus *et al.* 2016b) that is an important addition to reduce redundancies in the network model. Redundancy means here that two network units overlap so much that they are approximating almost the same partition of the input-output-space. For finding possible merge candidates, the grade of overlap

between units has to be evaluated over the input and output space. Similar to a merge approach discussed by Hennig (2010), I use the Bhattacharyya Coefficient (BC) to measure the overlap between two multivariate Gaussian distributions $G_1(\mu_1, \Sigma_1)$ and $G_2(\mu_2, \Sigma_2)$. BC has the advantage that the merge mechanism can be applied online based on currently available information of each unit, which is for example not possible for the merge mechanism in Ueda *et al.* (2000). In case of Gaussian distributions, a closed form solution for BC exists with

$$d_B(G_1, G_2) = \frac{1}{8} \cdot (\mu_1 - \mu_2)' \Sigma^{-1} (\mu_1 - \mu_2) + \frac{1}{2} \cdot log \frac{|\Sigma|}{\sqrt{|\Sigma_1| \cdot |\Sigma_2|}} \quad (3.13)$$

$$BC(G_1, G_2) = \exp\left(-d_B(G_1, G_2)\right) \quad (3.14)$$

Here, d_B is the Bhattacharyya distance and $\Sigma = (\Sigma_1 + \Sigma_2)/2$. For a similarity $S(i, j)$ between two units i and j, I have to calculate the overlap of the units's input and output probability density functions (pdf). Suppose, the input pdf of a unit i is $G_i^{input}(\mu_i, \Sigma_i)$ and the output pdf is $G_i^{output}(\tilde{W}_i \tilde{x}, \sigma_i^2 I)$. The similarity $S(i, j)$ is then calculated in regard to the overlap of the observed units for the input and output space

$$S(i, j) = BC(G_i^{input}, G_j^{input}) \cdot BC(G_i^{output}, G_j^{output}). \quad (3.15)$$

If $S(i, j) > T_{Merge}$ with a threshold T_{Merge}, then the units are possible merge candidates.

The merge mechanism starts merging always from the units with the maximal similarity. Its flow is therefore similar to a search that is repeatedly looking for the maximal similarity between units until no unit pair satisfies the merge threshold anymore. The algorithmic flow is described in the following:

1. Calculate the similarity $S(i, j)$ for all pairs $\{i, j\}$.
2. Choose the pair $\{i_{max}, j_{max}\}$ with maximal similarity.
3. If $S(i_{max}, j_{max}) > T_{Merge}$ then merge units into one and go to step 1.
4. Otherwise, stop routine.

The merge mechanism is computationally heavy, especially when the network model complexity M is high. Furthermore, it is unnecessary to apply merge at every time step t, because merge candidates are not found that frequently. Intervals of a few hundred time steps are sufficient. Yet, a new problem arises when applying merge in intervals, because the calculation

of the output BC depends on input x for the output center $\tilde{W}_i\tilde{x}$. A calculation of similarities using current input $x(t)$, in the form $\tilde{x}'(t) \equiv (x(t), 1)'$, is inappropriate since the calculated similarities depend on and change with $x(t)$. Preliminary experiments showed that this can lead to an underestimation of similarity, for example when $x(t)$ and the units are in different parts of the input space. A possible alternative would be to use the weighted sum $\langle\!\langle \tilde{x} \rangle\!\rangle'_i(t) \equiv (\langle\!\langle x \rangle\!\rangle_i(t), \langle\!\langle 1 \rangle\!\rangle_i(t))'$, however preliminary experiments showed that this approach is overestimating the similarity between the output distributions. Especially the first term of d_B becomes very small due to the output center. Therefore, I revise the overlap calculation to avoid the inclusion of input x in the output center for the BC calculation.

I use a multivariate theorem applicable to Gaussian distributions to conduct an affine transformation of the output distribution. According to the theorem, a distribution $U \sim N(\mu, \Sigma)$ can be linearly transformed with a vector c and a matrix D to a distribution $V \sim N(c + D\mu_U, D\Sigma_U D')$. Here, I want to transform the output distribution $y \sim N(\tilde{W}_i\tilde{x}, \sigma_i^2 I)$ so that input x is not included in the center of the output distribution. For convenience, I consider the transformation of the transpose $y' \sim N(\tilde{x}'\tilde{W}'_i, \sigma_i^2 I)$ instead. $\tilde{W}_i$ is defined in Eq. (2.19), and the transpose is

$$\tilde{W}'_i = (\langle\!\langle y\tilde{x}' \rangle\!\rangle_i(t)[\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t)]^{-1})' = [\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t)]^{-1}\langle\!\langle \tilde{x}y' \rangle\!\rangle_i(t). \tag{3.16}$$

I then use $U = y'$, $\mu_U = \tilde{x}'\tilde{W}'_i$, $\Sigma_U = \sigma^2 I$, $V = \tilde{W}'_i$, $D = [\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t)]^{-1}\langle\!\langle \tilde{x} \rangle\!\rangle_i(t)$ and $c = 0$ to transform U to V.

$$\mu_V = [\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t)]^{-1}\langle\!\langle \tilde{x} \rangle\!\rangle_i(t)\langle\!\langle \tilde{x}' \rangle\!\rangle_i(t)\tilde{W}'_i = \tilde{W}'_i \tag{3.17}$$

$$\Sigma_V = [\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t)]^{-1}\langle\!\langle \tilde{x} \rangle\!\rangle_i(t)\sigma_i^2 I([\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t)]^{-1}\langle\!\langle \tilde{x} \rangle\!\rangle_i(t))' = \sigma_i^2[\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t)]^{-1} \tag{3.18}$$

So, V becomes $\tilde{W}'_i \sim N(\tilde{W}'_i, \sigma_i^2[\langle\!\langle \tilde{x}\tilde{x}' \rangle\!\rangle_i(t)]^{-1})$, and input x is excluded from the output center. But $\tilde{W}'_i$ is a $(N+1) \times D$-dimensional matrix, and the left term of Eq. (3.13) becomes a $D \times D$-dimensional matrix dependent on the output dimension D. Therefore, I update Eq. (3.13) to

$$d_B(G_1, G_2) = \frac{1}{8D} \cdot Tr\left((\mu_1 - \mu_2)'\Sigma^{-1}(\mu_1 - \mu_2)\right) + \frac{1}{2} \cdot log\frac{|\Sigma|}{\sqrt{|\Sigma_1| \cdot |\Sigma_2|}}, \tag{3.19}$$

where Tr is the trace of the matrix.

Finally, I have to merge units i and j when $S(i_{max}, j_{max}) > T_{Merge}$. Again, I consider the units' Gaussian distributions for the input and output

space and merge the centers and covariances of input and output distributions in the same matter. New centers μ_{new} and covariances Σ_{new} are calculated by

$$\mu_{new} = \omega_i \mu_i + \omega_j \mu_j, \tag{3.20}$$

$$\Sigma_{new} = \sum_{k}^{i,j} \omega_k \left(\Sigma_k + (\mu_k - \mu_{new})(\mu_k - \mu_{new})' \right), \tag{3.21}$$

where $\omega_k = \frac{\langle\!\langle 1 \rangle\!\rangle_k(t)}{\langle\!\langle 1 \rangle\!\rangle_i(t) + \langle\!\langle 1 \rangle\!\rangle_j(t)}$ with $k \in \{i, j\}$ are functioning as merging weights with $\langle\!\langle 1 \rangle\!\rangle_k(t)$ representing the importance of unit i and j during training until current time step t. In addition, I initialize the newly merged unit's weighted sum of one to $\langle\!\langle 1 \rangle\!\rangle_{new}(t) = \langle\!\langle 1 \rangle\!\rangle_i(t) + \langle\!\langle 1 \rangle\!\rangle_j(t)$. Then, the remaining weighted sums can be initialized based on the available information.

3.3.2 Self-Constructing Model Adaptation

In online learning schemes, domain knowledge is generally limited what makes it difficult to decide on a good model initialization in advance. An alternative approach is to build the model from scratch during learning, and this approach has been applied several times in the context of receptive field based ANNs (Schaal and Atkeson 1998; Platt 1991). I introduce a similar approach in this thesis and build the network model of the NGnet self-constructively during learning with the help of the above discussed unit manipulation mechanisms.

Here, the NGnet is initialized with zero units ($M = 0$), and an initial unit is produced upon receiving the first training data sample. Only for the first unit, some predetermined initialization is necessary and I set the parameters as in the following.

$$\mu_1 = x(t) \tag{3.22}$$

$$\Sigma_1^{-1} = 0.1 I_N \tag{3.23}$$

$$\sigma_1^2 = 0.1 \tag{3.24}$$

$$\tilde{W}_1 = (0, y(t)) \tag{3.25}$$

After the first unit is produced, it is possible to create all other units in relation to the first one by applying the produce mechanism in Eq. (2.41)-(2.44). This reduces the necessary decisions for initialization largely, which eases the application of NGnets in environments where domain knowledge is limited.

The full flow of the self-constructing model adaptation is described in the algorithm below.

if M==0 **then**
  Produce first unit
else
  Compute Probabilities
  if $\sum_{i=1}^{M} P(x(t), y(t)|i, \theta(t-1)) < T_{Produce}$ **then**
    Produce new unit
  else
    Update units
    Test delete, split, merge
  end if
end if

Chapter 4

Experiments

In this chapter, I conduct several experiments to evaluate the learning performance of my proposed method for a broad range of learning tasks. First, two function approximation tasks are considered, where I do not only test the overall learning performance but also conduct experiments to discuss some of the new proposals independently. Additionally, the proposed method is evaluated for chaotic time series forecasting and a reinforcement learning tasks.

4.1 Preparations

Before the experiments, it is necessary to discuss several initial decision steps. This includes information about the compared NGnet learning methods and the choice of several learning parameters for the dynamic model selection and the discount schedules.

4.1.1 Compared Learning Methods

The learning performance of the proposed method is mainly evaluated in comparison with the previously proposed NGnet training methods (Celaya and Agostini 2015; Sato and Ishii 2000). The newly proposed method employs the changes described in Section 3.1 and it will be mainly termed either *LF* or *LF(Prop.)* in the following to keep things short. *LF* is applied with or without dynamic model selection depending on the aim of each experiment and the compared learning methods.

The previously proposed localized forgetting method (Celaya and Agostini 2015), *LF(Prev.)*, is mainly discussed in regard to *LF(Prop.)*. Dynamic model selection was yet to be considered for *LF(Prev.)* and it is not applicable without proposing some changes in the delete manipulation. Therefore, the two methods are compared only with static model complexities. To keep the comparison fair, both methods use an identical network initialization.

In addition, *LF (Prop.)* is compared with an NGnet that applies global forgetting updates, termed *GF* in the following. *GF* has been proposed in

Sato and Ishii (2000) and dynamic model selection has also been applied. Therefore, dynamic model selection can be included for the comparison of *LF (Prop.)* with *GF*. Two different versions of *GF* are considered: one is equal to the proposal in Sato and Ishii (2000) including a normalization coefficient as in Eq. (2.24), referred to as *GFnorm*, while the other one employs only a discount factor as in Eq. (2.26), referred to as *GFdisc*. For *GFdisc*, a normalizer is necessary to enable the application of a delete mechanisms for the same reasons as described in Section 3.3.1. Since old information is forgotten globally, units experience some major changes in the parameters at every time step. Therefore, the applied normalizer equals a unit life time counter, which is the same as the second normalizer candidate discussed in Section 3.3.1. For both *GF* methods, I apply dynamic model selection in a self-constructing manner in all experiments as explained in Section 3.3.2.

Finally, some differences have to be considered for the initialization of the learning approaches. *GFdisc* and both local forgetting methods are based on a non-normalized update method that accumulates information about the received training samples in weighted sums. On the other hand, *GFnorm* is using a normalizer that accumulates newly received information in weighted means. This makes it necessary to consider a slightly different initialization for *GFnorm* to achieve a good and approximately similar learning behavior compared with *GFdisc*. For the non-normalized update methods, the weighted accumulator $\langle\!\langle 1 \rangle\!\rangle_{new}$ is initialized with one for a newly added unit (with dynamic model selection) or units created at the beginning (no dynamic model selection). This means a unit has a representation ability of one for its input center at the point of its creation. Yet, this initialization would put too much weight on a newly added unit in case of *GFnorm*, where the weight mean of one is supposed to be normalized and then is equal to $\eta(t)P_i(t)$. Therefore, I will use the initialization $\langle\!\langle 1 \rangle\!\rangle_{new} = \eta(t)$ for *GFnorm* instead. With this additional change in the initialization of *GFnorm*, it is then possible to compare *LF* with the global forgetting approaches *GFdisc* and *GFnorm* under equal initial conditions.

4.1.2 Model Selection Parameter Settings

Each of the four dynamic model selection mechanisms include a threshold parameter that has to be set in advance, where the four manipulation mechanisms are produce (*Prod.*), delete (*Del.*), split (*Spl.*) and merge (*Mrg.*). I have conducted several preliminary experiments with different parameter settings for all compared methods and the best performing parameters were selected separately for each test environment and method. An overview of the selected parameters can be found in Table 4.1, where I have

TABLE 4.1: Manipulation Parameter Settings for All Experiments

Method	Parameter Settings			
	Prod.	Del.	Spl.	Mrg.
FA: Balanced				
LF(Prop.)	1.0	0.09	3×	0.7
GF	1.0	0.001	0.12	-
FA: Imbalanced				
LF(Prop.)	1.0	0.001	3×	0.95
GF	1.0	0.00001	0.12	-
FA: Dynamic				
LF(Prop.)	1.0	0.05	3×	0.8
GF	1.0	0.0001	0.12	-
Lorenz Attractor				
LF(Prop.)	0.001	0.01	3×	0.7
GFdisc	0.001	0.0001	10×	-
MG Time Series: 6-steps ahead				
LF(Prop.)	500.0	0.1	3×	0.7
GFdisc	500.0	0.001	3×	-
MG Time Series: 50-steps ahead				
LF(Prop.)	500.0	0.01	3×	0.7
GFdisc	500.0	0.00001	5×	-
Reinforcement Learning				
LF(Prop.)	0.00001	0.001	3×	0.7
GFdisc	0.00001	0.00001	2	-

summarized the two global forgetting methods as *GF* when the same parameters have been selected after extensive testing. For the function approximation task, several different learning scenarios are tested that are all marked with a beginning *FA* in the table. Some split parameters in the table are a multiple of a referenced value and are marked by an additional $\times$, e.g. $3\times$ meaning three times. For the chaotic time series forecasting tasks, I also apply a dynamic threshold decision for *GFdisc*'s split mechanism, because the static one used in Sato and Ishii (2000) has unstable performance. A unit is then split when its output variance is a certain number of times bigger than the average of the units' variances.

The produce threshold is set to the same parameter for all compared methods within an experimental environment. This is reasonable because the same produce mechanism is used by all methods. In the experiments, differences in the unit production behavior can be observed, but this is mainly caused by the effects of the different forgetting approaches and delete mechanisms. After testing several values, the produce threshold is set to $T_{Produce} = 1.0$ for the *FA* experiments and $T_{Produce} = 0.001$ for the

Lorenz Attractor experiment. For the *Mackey-Glass (MG) Time Series* experiment, very high produce thresholds are chosen to make the NGnet competitive. Further explanation of the reasons can be found in Section 4.3.3. For the reinforcement learning task, it is however necessary to choose very small produce thresholds.

4.1.3 Scheduling of Discount Factor

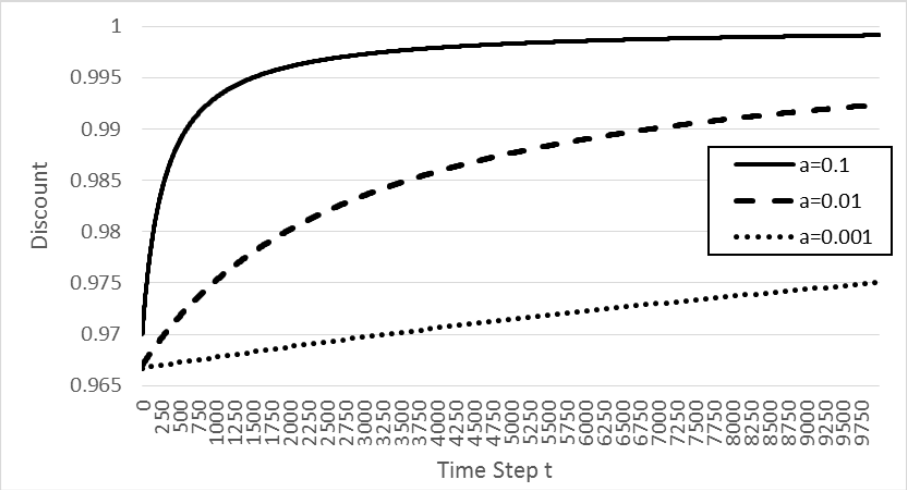


FIGURE 4.1: Discount Scheduling for b=30

A discount factor $\lambda(t)$ has been introduced to the NGnet updates with the aim to speed up convergence. To reach convergence, the discount has to be scheduled so that $\lambda(t) \to 1$ when $t \to \infty$ to fulfill the Robbins-Monro condition for convergence of stochastic approximations (Kushner and Yin 1997). A scheduling $\lambda(t) \to 1$, where $\lambda(t)$ is changing over time t, can be calculated according to

$$\lambda(t) = 1 - \frac{1-a}{at+b}, \tag{4.1}$$

depending on two parameters a and b. a controls how fast $\lambda(t) \to 1$ and b sets the initial value of λ. In Fig. 4.1, it is shown how different values of a influence the convergence of $\lambda(t)$ for an example where the initial value is $b = 30$. In my experiments, I apply several values for a and b to evaluate the effect of different discount schedules on the performance of all compared methods.

In addition to experiments with forgetting, I conduct experiments without forgetting. Here, a discount factor $\lambda(t) = 1$ is applied over the whole training time which is equal to any value b and $a = 1$. Then, *LF* updates with Eq. (3.11) reduce to the same as *GFdisc*'s with Eq. (2.26). This makes a

better comparison of the two different model selection approaches possible, since learning performance is not influenced by differences in the update methods.

4.2 Function Approximation Tasks

First of all, the NGnets are tested for two function approximation tasks. Here, the main focus is laid on the second function approximation task, described in Section 4.2.2, but in addition I consider another simple approximation task in Section 4.2.1 to discuss the effects of different update precisions in a second testbed.

4.2.1 Simple Regression Task with 5-Dimensions

The main aim of this simple approximation task is to investigate the effect of different update precisions on the learning performance of *LF*. In section 3.2, I have discussed that it is not necessary to update all units at every time step when an NGnet is updated with local forgetting, because units with small update weights have barely any influence on the learning performance. It is however difficult to decide on an appropriate update precision without any experimental results as references, so I use the following experiment as a baseline for the decision. The same function approximation task has been applied earlier by Jacobs (1997) to analyze the bias and variance of different ME model architectures. In a similar manner, I want to investigate the changes in learning performance, bias and variance of NGnets that employ different update precisions. The results are then used to decide on a limit for the update precision that ensures stable learning of the NGnet even under changing environmental conditions.

For this learning task, the target function is defined as

$$f(\mathbf{x}) = \frac{1}{13}\left[10sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5\right] - 1, \qquad (4.2)$$

where $\mathbf{x} = [x_1, \ldots, x_5]$ is a 5-dimensional input vector and each input component can take values between zero and one. The target function $f(x)$ takes then values in the interval $[-1, 1]$. Additionally, some white Gaussian noise $N(0, \sigma^2)$ is added to the target outputs.

Preparations

Some preparative steps are necessary before the experiment, which are explained in the following. This includes details about the training and test

data, the network initialization and the performance evaluation. In addition, a discount schedule with $a = 0.001$ and $b = 30$ is used for all test cases.

Training and Test Data For this approximation tasks, I prepare 25 training data sets with 500 input-output patterns each. The input data are sampled i.i.d. from a uniform distribution over the full interval of each input component $[0, 1]$. Then, the target output is calculated using $f(x)$ and an additional noise is added. Because I want to evaluate different noise environments, I create three output data sets where the Gaussian noise variances are $\{0.2, 0.1, 0.05\}$. All output data sets corresponds to the same 25 input data sets. In addition, one test data set is created, where input data are not sampled randomly, but 1024 samples are uniformly spaced in the 5-dimensional input space instead. The target outputs are not corrupted by noise to make the calculation of bias and variances possible. The same test data set is applied to all training runs.

Network Initialization To ensure that differences in performance are solely due to the different training data sets, the same network initialization is used for all test cases. Only NGnets with a static model complexity are tested, where two representative model complexities are used with 3 and 8 units. The lower model complexity has been found to perform best after testing different complexities in the range from 2 to 10. The higher model complexity then represents a case with unnecessarily high complexity. The input centers are initialized with the same value for each dimension, where the value is obtained by equally spacing the total number of units in the interval from zero to one. Similarly, the weight matrix of each unit is initialized with zeros for all columns except the last, where initial values are obtained by equally spacing the number of units in the full output interval $[-1, 1]$.

Learning Performance The overall learning performance is evaluated for the test data set with the Mean Square Error (MSE) after 60 learning epochs. The MSE is defined by

$$MSE = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{S}\sum_{s=1}^{S}|\hat{f}(x(t))^{(s)} - y(t)|^2, \tag{4.3}$$

where T is the number of training data, S is the number of simulation runs, which is equal to the number of training data sets, and $\hat{f}(x(t))^{(s)}$ is the estimated output of the NGnet for the t-th data sample in the s-th simulation run.

Bias and Variance In addition to the learning performance, the bias and variance are two other quantities of interest that have been already shortly mentioned in Section 2.2.2 in regard to the model complexity selection problem. The bias-variance dilemma also plays a role in regard to the global-local trade-off of a network. Local architectures are known to over-fit more easily resulting in high variances. For the ME framework, it has been claimed by Jordan and Jacobs (1994) that when a network model splits the input space hardly between its units, it has a larger variance as when it splits the input space softly. This was also experimentally tested and confirmed in Jacobs (1997). Bias and variance play therefore a role in the update precision evaluation, since lower update precisions resemble a harder split of the input space as a smaller number of units is updated.

A bias is a quantity that measures the difference between the target output and the average estimated output of network models, which are trained on different training sets, for a data sample at time step t. The bias is calculated with

$$Bias = \frac{1}{T}\sum_{t=1}^{T} |\overline{\hat{f}(x(t))} - y(t)|^2, \tag{4.4}$$

where $\overline{\hat{f}(x(t))}$ is the average output of different simulations for a sample at time step t and is defined by

$$\overline{\hat{f}(x(t))} = \frac{1}{S}\sum_{s=1}^{S} \hat{f}(x(t))^{(s)}. \tag{4.5}$$

The variance is a quantity that measures how much the estimated output of each processing units in the separately trained network models varies around the average estimation of each unit. It is calculated with

$$Variance = \sum_{i=1}^{M}\frac{1}{T}\sum_{t=1}^{T}\frac{1}{S}\sum_{s=1}^{S}(\hat{f}_i(x(t))^{(s)} - \overline{\hat{f}_i(x(t))})^2. \tag{4.6}$$

Here, $\hat{f}_i(x(t)) = \tilde{W}_i\tilde{x(t)}N_i(x(t))$ is the i-th unit output for an input data sample $x(t)$ at each simulation and $\overline{\hat{f}_i(x(t))}$ is the average output of i-th unit overall simulation that is defined by

$$\overline{\hat{f}_i(x(t))} = \frac{1}{S}\sum_{s=1}^{S} \hat{f}_i(x(t))^{(s)}. \tag{4.7}$$

Experimental Results

Experimental results are presented separately on three tables with one table for each performance measure: MSE (Table 4.2), bias (4.3) and variance

TABLE 4.2: MSE Results for Simple Regression Task

Update Precision	UnitNo=3			UnitNo=8		
	Var0.2	Var0.1	Var0.05	Var0.2	Var0.1	Var0.05
1.0E-17	**0.0168**	**0.0117**	**0.0101**	0.0351	0.0184	0.0115
1.0E-7	**0.0168**	**0.0117**	**0.0101**	0.0351	0.0184	0.0115
1.0E-6	**0.0168**	**0.0117**	**0.0101**	0.0349	0.0185	0.0115
1.0E-5	**0.0168**	**0.0117**	**0.0101**	**0.0345**	**0.0181**	0.0114
1.0E-4	**0.0168**	**0.0117**	**0.0101**	0.0358	0.0185	0.0115
1.0E-3	0.0169	**0.0117**	**0.0101**	0.0374	0.0184	**0.0103**
1.0E-2	0.0179	0.0118	0.0105	0.0359	0.0183	0.0116
1.0E-1	0.0193	0.0132	0.0115	0.0407	0.0213	0.0129

(Table 4.4). In the first column of each table, the tested update precision is listed, where the stated values are the limits of the included update weights $P_i(t)$ so that a unit is updated only when $P_i(t) > 1.0E-XX$. I don't display update precisions in the range from $1.0E-16$ to $1.0E-8$, because there is no obvious difference compared with the update precision $1.0E-17$. In all tables, experimental results for several test cases are presented, which employ different update precisions, noise levels and number of units.

The most general performance measure is the MSE (Table 4.2), which gives an overview of how well each test case generalizes on the test data set. In the table, the best learning performance is marked in for each column in bold. The results show that a lower update precision is not necessarily always performing worse, but there are cases where learning performance improved instead. Yet, it is difficult to find a common pattern in the learning results that would give a hint on the likely best performing update precision. When the model complexity is low (*UnitNo=3*), higher precisions perform generally better than lower ones. There is no real difference in the learning performance for precisions higher than $1.0E-3$, while the learning performance starts to worsen for precisions under this limit. On the other hand, bigger differences can be observed for the tested precisions when the model complexity is relatively high (*UnitNo=8*). Interestingly, the best learning performance is now achieved somewhere in the middle range, where additional differences can be observed for the changing noise levels. Since the higher model complexity actually represents a case of over-fitting, performance is worse than for the lower model complexity, which is especially apparent for the highest noise level (variance $\sigma^2 = 0.2$). Overall, a decrease in learning performance can be observed for the lowest update precision $1.0E-1$ for all test cases. This is likely because of the reasons that have been discussed already in Section 3.2.

Additional insights for the above discussed MSE results can be found in

TABLE 4.3: Bias Results for Simple Regression Task

Update Precision	UnitNo=3			UnitNo=8		
	Var0.2	Var0.1	Var0.05	Var0.2	Var0.1	Var0.05
1.0E-17	0.0037	0.0025	0.0023	0.0021	0.0013	0.0010
1.0E-7	0.0037	0.0025	0.0023	0.0021	0.0013	0.0010
1.0E-6	0.0037	0.0025	0.0023	0.0020	0.0013	0.0010
1.0E-5	0.0037	0.0025	0.0023	**0.0019**	0.0012	0.0010
1.0E-4	0.0037	0.0025	0.0023	0.0021	0.0012	0.0009
1.0E-3	0.0037	0.0025	0.0023	0.0022	0.0011	0.0009
1.0E-2	0.0037	**0.0022**	0.0028	0.0025	**0.0010**	**0.0008**
1.0E-1	0.0039	0.0024	**0.0020**	0.0025	0.0011	0.0011

TABLE 4.4: Variance Results for Simple Regression Task

Update Precision	UnitNo=3			UnitNo=8		
	Var0.2	Var0.1	Var0.05	Var0.2	Var0.1	Var0.05
1.0E-17	0.0462	**0.0385**	**0.0382**	0.0785	0.0576	0.0520
1.0E-7	0.0462	**0.0385**	**0.0382**	0.0785	**0.0575**	0.0520
1.0E-6	0.0462	**0.0385**	**0.0382**	**0.0781**	0.0576	0.0520
1.0E-5	0.0462	**0.0385**	**0.0382**	0.0791	0.0578	0.0521
1.0E-4	0.0462	**0.0385**	**0.0382**	0.0794	0.0584	0.0527
1.0E-3	**0.0460**	**0.0385**	0.0383	0.0833	0.0595	**0.0513**
1.0E-2	0.0481	0.0393	0.0384	0.0818	0.0617	0.0535
1.0E-1	0.0518	0.0416	0.0425	0.0930	0.0683	0.0588

the bias (4.3) and variance (Table 4.4) tables. Again, I marked the best performing (lowest) values for each column in bold. One crucial performance factor is to find a good balance between bias and variance, where a high variance implies often a low bias (over-fitting) and a low variance implies generally a high bias (under-fitting). One obvious example of over-fitting is presented here with the high model complexity (*UnitNo=8*) testbed. While MSE results showed that the *UnitNo=8* testbed performs worse than the low model complexity testbed, it has a lower bias. On the other hand, the variance is much higher meaning that the network model starts to fit noise and therefore results in the worse performance. Furthermore, it is observable that a lower update precision is not necessarily related to a worse bias and changes in the bias are overall relatively low in most cases. But the variance becomes higher for lower update precisions in all cases. Therefore, I conclude that too low update precisions are prone to over-fit and should be avoided. The question is then how to choose an update precision that has no greater negative influence on the learning performance but can lower the computational complexity as much as possible. In cases where model complexity is near to optimum, it does not seem like a problem to choose very low complexities as $1.0E-3$. Yet, it is not possible to assume an always near

optimum model complexity when considering dynamic model selection, especially for self-constructing training methods. The decision on an appropriate update precision is difficult with only one testbed, especially because the over-fitting model complexity testbed shows high fluctuations in the learning performance. For now, I consider $\{1.0E-6, 1.0E-5, 1.0E-4\}$ as possible candidates for the update precision and conduct another experiment to evaluate the learning performance in an additional test environment (Section 4.2.2).

4.2.2 The Cross Function

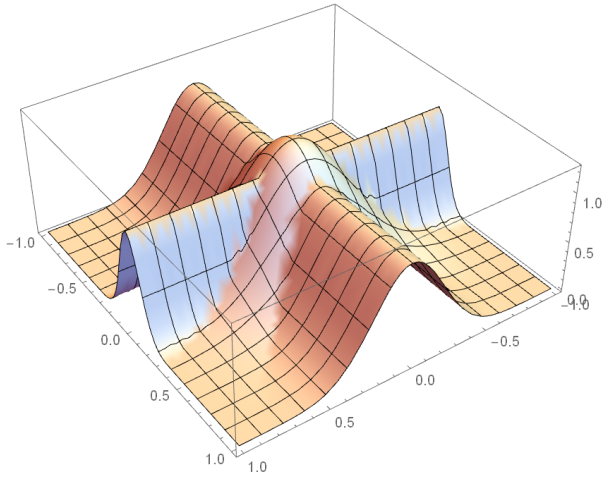


FIGURE 4.2: Target of the Cross Function

In the following section, I consider a commonly used function approximation task of the *Cross function* (Celaya and Agostini 2015; Meier *et al.* 2014; Vijayakumar *et al.* 2005; Sato and Ishii 2000; Schaal and Atkeson 1998), which provides a learning tasks that is sufficiently complex but still can be illustrated nicely. Therefore, it makes a good testbed to evaluate not only the overall learning performance but also conduct performance test for partial improvements of the proposed method separately. The function has the input dimension $N = 2$, the output dimension $D = 1$, and is defined by

$$g(x_1, x_2) = \max\{e^{-10x_1^2}, e^{-50x_2^2}, 1.25e^{-5(x_1^2+x_2^2)}\}. \tag{4.8}$$

In addition, a normally distributed random noise $\epsilon(t) \sim N(0, 0.01)$ is added to each function output $g(x_1(t), x_2(t))$, and $y(t) = g(x_1(t), x_2(t)) + \epsilon(t)$ is obtained as the noisy sample output. The function possesses areas with rather high and rather low curvature and a Gaussian bump in the middle (see Fig. 4.2). It is challenging to find a good approximation, since the nonlinearities

are hard to capture accurately when model complexity is too low, but models tend to overfit easily for higher model complexities (Vijayakumar *et al.* 2005).

Preparations

Again, several preparative steps are discussed that were taken before training the NGnets. This includes topics about the used training and test data sets, different performance evaluation criteria and the static network initialization.

Training Data For this learning task, different input data distributions are used to evaluate the robustness of the compared methods in a variety of learning scenarios. When data are not i.i.d., neural networks are more prone to negative interference and achieving good performance is difficult. Three test cases are considered

- *Balanced*
- *Imbalanced*
- *Dynamic*

For the *balanced* test case, the training data are i.i.d over the whole input space $(-1 \leq x_1, x_2 \leq 1)$. For the *imbalanced* test case, a non-identical data distribution is used. Here, 95% of the data samples are extracted from a sub-region of the input domain with $(0 \leq x_1, x_2 \leq 0.25)$, and the remaining data are i.i.d in $(-1 \leq x_1, x_2 \leq 1)$. For these two cases, 10,000 data samples are obtained for each test run. For the *dynamic* test case, the input data distribution is slowly changing for x_1 over time. x_1's sample distribution starts at the interval $[-1, -0.2]$ and ends at the interval $[0.2, 1]$ after 250,000 training samples. x_2 is sampled in $[-1, 1]$ over all time steps t. A similar experiment has been conducted by Celaya and Agostini (2015) and Sato and Ishii (2000) to evaluate learning performance in dynamic environments. In all test cases, I have obtained training data sets for 50 test runs, and the same data sets are applied to all compared methods.

Test Data Similarly as in Celaya and Agostini (2015), test data are obtained from a regular grid formed of 21×21 points in the input domain. These data are used to evaluate the learning performance of the compared methods over the whole input space in all three test cases. Additionally, test data is obtained for the last input interval of the *dynamic* test case. Here, performance is evaluated with two test sets representing the whole input space and the last training interval, which are referred to as *NMSE All* and *NMSE Last* in the result tables respectively.

Performance Evaluation For all Cross function experiments, the learning performance of the tested methods is evaluated with the normalized Mean Square Error (NMSE). The NMSE is defined by

$$NMSE = \frac{\sum_{t=1}^{T}(y(t) - \hat{f}(x(t)))^2}{\sum_{t=1}^{T}(y(t) - \bar{y})^2}, \tag{4.9}$$

where T is the number of test data samples and the divisor is equal to the variance of the target test data. In the result tables, all NMSEs are presented as the averages of the 50 test runs.

Permutation Tests Additionally, one-tailed permutation tests are conducted. Since only the averages over 50 test runs are presented as results, p-values, which are obtained by the permutation test, provide an additional measure to evaluate the significance of the compared methods' performance differences. Generally, *LF* is compared with one of the other tested methods for all presented p-values in the result section.

Network Initialization When dynamic model selection is considered, the NGnet is initialized with zero units. Otherwise, the network model has to be initialized in advance and I consider two static model complexity initializations for this learning task. First, a model complexity with 25 units is used, which is equal to the complexity used by Celaya and Agostini (2015). Additionally, I add some experiments with a model complexity of 100 units. For both cases, units are placed equally spaced on a grid of the input space. During the initialization, (x, y) values are obtained according to the location of each unit on the grid, and the unit's parameters are then set according to Eq. (3.22) - (3.25).

Comparison of Localized Forgetting Methods

In this experiment, I compare my proposed method *LF (Prop.)* with the former local forgetting method (*LF (Prev.)*, Celaya and Agostini (2015)) for the *Balanced* testbed. This experiment evaluates the effectiveness of the *LF (Prop.)* update function (Eq. (3.11)) in regard to the previous one (Eq. (2.32)) and discusses similarities and differences. The two methods basically differ only in the applied update weight, and the new local forgetting approach was mainly proposed to eliminate the inability of *LF (Prev.)* to update network parameters over the whole numerical range of discount factor $\lambda(t)$. For *LF (Prev.)*, $\lambda(t) = 1$ cannot be processed because the update weight becomes $\frac{0}{0}$ in this case. Because dynamic model selection has not been considered for *LF (Prev.)* and also to keep the comparison simple, testing is conducted only for the two static model complexities. Performance results are

TABLE 4.5: Comparison of Localized Forgetting Methods

Discount	Unit No. = 25			Unit No. = 100		
	NMSE		p-Val	NMSE		p-Val
	LF(Prop.)	LF(Prev.)		LF(Prop.)	LF(Prev.)	
b=10						
a=0.1	**0.0386**	0.0394	0.01	**0.0246**	0.0247	0.25
a=0.01	**0.0244**	0.0251	0.01	**0.0185**	0.0186	0.16
a=0.001	**0.0170**	0.0179	0.10	**0.0109**	0.0111	0.08
b=30						
a=0.1	**0.0398**	0.0402	0.01	0.0252	**0.0251**	0.29
a=0.01	**0.0290**	0.0293	0.03	0.0215	0.0215	0.20
a=0.001	**0.0235**	0.0237	0.14	0.0184	**0.0183**	0.19
b=150						
a=0.1	**0.0420**	0.0421	0.02	0.0259	0.0259	0.33
a=0.01	**0.0380**	0.0381	0.01	0.0249	0.0249	0.30
a=0.001	**0.0367**	0.0368	0.01	**0.0245**	0.0246	0.30
b=1000						
a=0.1	0.0442	0.0442	0.00	0.0264	0.0264	0.38
a=0.01	0.0438	0.0438	0.00	0.0263	0.0263	0.40
a=0.001	0.0437	0.0437	0.00	0.0263	0.0263	0.40
Discount 1						
a=1.0	**0.0451**	0.3650	0.00	**0.0267**	0.3786	0.00

presented in Table 4.5 with the two model complexities *Unit No. = 25* and *Unit No. = 100*. When one method performed apparently better, experimental results are marked in bold. I have obtained test results for 13 different discount schedules with four different initial values $b = \{10, 30, 150, 1000\}$ and four different convergence schedules $a = \{1, 0.1, 0.01, 0.001\}$. In case of $a = 1$, the value of b is irrelevant as the discount $\lambda(t)$ becomes one over all t and no forgetting occurs. In the other 12 cases, the applied discount factor plays a role in the resulting learning performance.

Overall, there is no big differences between the two approaches in most cases when forgetting occurs. Yet, there is a small tendency for *LF(Prop.)* to perform better than *LF(Prev.)*. Interestingly, the differences is more apparent for the low model complexity with 25 units, where *LF(Prop.)* performs always better or equally well. The performance difference becomes much smaller for the higher model complexity with 100 units and in two cases *LF(Prev.)* is able to perform slightly better. Also, the number of equally well-performing cases has increased. The largest performance difference is observed for the discount initial value $b = 10$ for both model complexities. Again, the difference is much smaller for *Unit No. = 100*, but *LF(Prop.)* is still able to perform slightly better. These results lead to the assumption that the influence of the update weights becomes smaller with increasing model

complexity. Furthermore, an observable tendency is that the difference between the two methods becomes bigger with smaller b, which is equal to an initial discount values farer from one. Fig. 4.3 shows an example of how the learning performance is changing over time t for the compared methods *LF(Prop.)* and *LF(Prev.)* in the test case {*Unit No.* = 25, $a = 0.1$, $b = 10$}. Although the difference is small, *LF(Prop.)* is able to perform better than *LF(Prev.)* over the whole learning time.

The obtained p-values also show that the superiority of *LF (Prop.)* for the 12 forgetting cases cannot be claimed in all cases. Especially for the high model complexity with 100 units, the p-values are high meaning that the significance of performance differences is low. Yet, when model complexity is small with 25 units, *LF(Prop.)* is able to perform significantly better in most cases even when the approximate learning performance is quite similar. A special case can be observed for $b = 1000$, where the approximate learning performance is the same but the significance of performance differences is high, represented by a p-value that is approximately zero. For all three $b = 1000$ discount schedules, *LF(Prop.)* is performing slightly better but the performance results are outside of the presented precision limits with an approximately $7.0E - 07$ difference. Still, the performance difference is steady over most of the 50 test runs, and *LF(Prop.)* is therefore able to perform better with high significance. The p-value comparison again confirms that a higher model complexity results in a smaller performance gap between the two methods and also in a lower significance. Yet, the overall tendency shows that *LF(Prop.)* is favorable over *LF(Prev.)*.

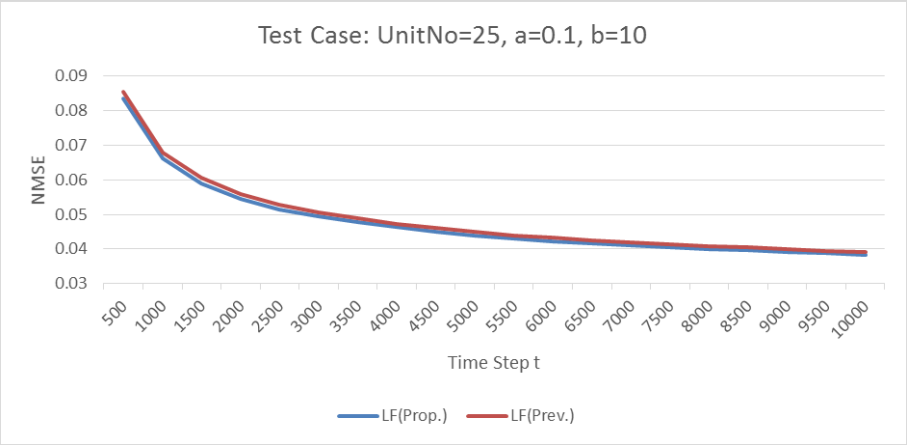


FIGURE 4.3: Comparison of Local Forgetting Methods' Performance Error over Learning Time

For the results without forgetting (*Discount 1*), *LF (Prop.)* performs much better than *LF(Prev.)* for both network sizes. The p-values also prove the

TABLE 4.6: Update Precision Test for Cross Function with Noise Variance 0.01 ($b = 30$)

Update Precision	Discount1		a=0.01		a=0.001		a=0.0	
	NMSE	UnitNo	NMSE	UnitNo	NMSE	UnitNo	NMSE	UnitNo
1.0E-17	0.0316	52.24	0.0208	60.06	0.0157	65.02	0.0152	66.49
1.0E-7	0.0316	52.24	0.0208	60.06	0.0157	65.02	0.0152	66.49
1.0E-6	0.0316	52.24	0.0208	60.06	0.0157	65.02	0.0152	66.49
1.0E-5	0.0316	52.24	0.0208	60.08	0.0157	65.02	0.0152	66.51
1.0E-4	0.0317	52.29	0.0207	60.08	0.0158	65.00	0.0151	66.55
1.0E-3	0.0317	52.27	0.0205	60.35	0.0155	65.35	0.0149	67.10
1.0E-2	0.0284	53.51	0.0181	63.12	0.0149	68.71	**0.0141**	70.22
1.0E-1	**0.0204**	61.86	**0.0160**	75.06	**0.0147**	82.53	0.0145	84.39

TABLE 4.7: Update Precision Test for Cross Function with Noise Variance 0.1 ($b = 30$)

Update Precision	Discount1		a=0.01		a=0.001		a=0.0	
	NMSE	UnitNo	NMSE	UnitNo	NMSE	UnitNo	NMSE	UnitNo
1.0E-17	0.0497	124.61	0.0501	132.55	**0.0508**	136.94	**0.0514**	138.10
1.0E-7	0.0497	124.61	**0.0500**	132.55	**0.0508**	136.94	**0.0514**	138.10
1.0E-6	0.0497	124.61	**0.0500**	132.55	**0.0508**	136.94	**0.0514**	138.10
1.0E-5	0.0497	124.61	**0.0500**	132.57	0.0509	136.98	**0.0514**	138.10
1.0E-4	0.0497	124.76	0.0502	132.65	0.0510	137.08	0.0517	138.27
1.0E-3	**0.0495**	125.31	0.0508	133.55	0.0522	138.22	0.0523	139.43
1.0E-2	0.0525	129.82	0.0536	138.61	0.0546	143.61	0.0551	144.88
1.0E-1	0.0799	150.96	0.0783	161.98	0.0824	168.76	0.0833	170.57

superiority of my proposal. Here, *LF(Prev.)* was not able to update its network parameters, and its performance is the direct result of the initialization. Therefore, I have achieved the main purpose of the new proposal. Now, updates are possible for the local forgetting approach, even when the discount equals one. This complies with the numerical range defined for the discount factor as $0 \leq \lambda \leq 1$ by Sato and Ishii (2000).

Discussing the Update Precision

In addition to Section 4.2.1, I want to further discuss the effects of different update precisions on the learning performance of the NGnet for the here applied Cross function. To add further insights, I make the test different from the simple regression task and only evaluate the learning performance of the trained NGnets but over a broader range of discount schedules with the initial value $b = 30$ and different convergence schedules $a = \{1, 0.01, 0.001, 0.0\}$. Also, I do not use a static model complexity but instead train the NGnets with a self-constructing model selection approach that only includes production of new units. This enables me to observe

learning performances of different update precisions in a setting with dynamic model selection that still is kept simple enough to make a fair comparison possible without influence from the other manipulation mechanisms. The experiments are conducted for two noise levels with the commonly applied variance 0.01 and a higher noise variance 0.1. Experimental results are presented separately for the two noise levels in Table 4.6 and Table 4.7 respectively. Again, the best performing results are marked in bold for each column.

For the results with noise variance 0.01 in Table 4.6, it can be observed that the lowest update precision $1.0E-1$ performs best in most cases. Yet, this comes at the cost of an increased model complexity, which is likely related to the units participating in a decreased range of input space. This leads then to smaller input covariances of the NGnet and the necessity to have more units to cover the input space accurately. The discount schedule also influences the best update precision performance. While discount schedules that converge to one faster tend to have the best performance for a $1.0E-1$ update precision, this is not the case when $a = 0.0$. Here, an update precision with $1.0E-2$ performs best, while $1.0E-1$'s performance is still better than update precisions with $1.0E-3$ or larger. Yet, this shows that higher discounts might negatively affect the learning performance when the update precision is too low.

The same tendencies can be observed more extremely in Table 4.7 for the higher noise level with variance 0.1. Here, the higher noise provokes the NGnet to over-fit more easily and the learning performance decreases drastically for the lower update precisions, especially $1.0E-1$, while model complexity also drastically increases. For a discount of one over all t, update precision $1.0E-3$ is able to perform best, but for all other discount schedules higher update precisions are preferable. This shows again that higher discounts affect the learning performance negatively when update precisions are too low. Therefore, even when very low update precisions can perform best in some test cases where the noise level is moderate, they have to be generally handled with care when information about the learning environment are limited. The observed proneness to noise is then similar to the insights obtained for the first testbed in Section 4.2.1.

In Section 4.2.1, I already assumed that the update precisions $\{1.0E-6, 1.0E-5, 1.0E-4\}$ are the best possible candidates to choose an appropriate update precision. For both noise levels, it is observable that there is almost no change in learning performance and model complexity over all test cases for update precisions with $1.0E-5$ or higher. For update precision $1.0E-4$, the performance and model complexity are still similar compared with the higher update precisions but differences become already more apparent. These observations are similar to the ones of Section 4.2.1, where

TABLE 4.8: Update Precision Test for Cross Function with Noise Variance 0.01 ($a = 0.0$)

Discount/ Precision	UnitNo. = 25		UnitNo. = 100	
	NMSE	Time(sec.)	NMSE	Time(sec.)
b=10				
All Units	0.0174	4.86	0.0074	16.59
1.0E-17	0.0174	2.89	0.0074	10.91
1.0E-5	0.0175	**2.15**	0.0074	**6.58**
b=150				
All Units	0.0399	4.81	0.0619	16.45
1.0E-17	0.0399	3.68	0.0619	13.46
1.0E-5	0.0399	**2.45**	0.0619	**7.60**
Discount=1				
All Units	0.0516	4.80	0.0700	16.44
1.0E-17	0.0516	3.80	0.0700	13.63
1.0E-5	0.0516	**2.50**	0.0699	**7.66**

update precision $1.0E-4$ also has difficulties to perform similarly well when the noise level is high. Therefore, I decide to set the update precision for the *LF(Prop.)* update method to $1.0E-5$.

After deciding on the new update precision, I want to evaluate the achieved difference in computational time in comparison with two representative update precisions. The results are presented in Table 4.8. Here, *All Units* refers to a training approach where all units are always updated regardless of the actual necessity of the update. For the other two cases, updates are limited to units that have update weights larger than $1.0E-17$ or $1.0E-5$, where the first one is the computational precision and the second is my selected precision. For a fair comparison, I use static model complexities with 25 and 100 units to see the time differences for a low complexity and higher complexity case, since the model complexity highly influences the learning speed. Time is measured in seconds (sec.) and the best learning speed for each test case is marked in bold in the table. The presented results are again the average of 50 test runs and only one test run was executed on one time to reduce outer influences as much as possible. The results in Table 4.8 show that the update precision $1.0E-5$ is able to improve the learning speed a lot while achieving approximately the same performance results as the other two update precisions. For the presented cases, the update precision $1.0E-5$ improves learning speed approximately 25% to 45% compared with $1.0E-17$ and 50% to 60% compared with an update of all units.

Balanced Testbed With Dynamic Model Selection

For this testbed, I compare the proposed *LF* method with the two global forgetting methods *GFdisc* and *GFnorm* for the Cross function with a balanced

TABLE 4.9: Balanced Test without Forgetting

Method	NMSE	UnitNo.	Manipulation Counter				p-Val
			Prod.	Del.	Spl.	Mrg.	
Setting 1							
LF(Prop.)	**0.0202**	45.06	08.74	50.96	2.46	6.18	
(LF(no Mrg.))	(0.0200)	(46.3)	(93.1)	(50.38)	(2.58)	(/)	
GFnorm	0.0382	30.9	94.52	67.12	2.5	/	0.0
GFdisc	0.0351	44.42	57.02	15.98	2.38	/	0.0
Setting 2							
LF(Prop.)	0.0301	49.06	57.14	0.42	3.22	11.88	
(LF(no Mrg.))	(0.0311)	(55.04)	(51.06)	(0.0)	(2.98)	(/)	
GFnorm	**0.0292**	54.2	50.6	0.04	2.64	/	0.08
GFdisc	**0.0292**	53.9	50.58	0.24	2.56	/	0.08

training data distribution. All methods apply a self-constructing dynamic model selection. The experimental results are presented separately for two test settings: without forgetting and with forgetting of previously learned information.

Without Forgetting First, the three methods are compared with each other for the *balanced* test case without forgetting. Experimental results are presented in Table 4.9 for two different manipulation parameter settings. The best performing results are marked in bold for each setting. For both settings, I have chosen the same parameters as stated in Table 4.1 for *FA: Balanced* except the delete threshold parameters. In addition, information are included about the number of manipulation occurrences, because they give interesting insights, especially in regard to the different deletion behavior of the three methods. Furthermore, experimental results for *LF(Prop.)* without merging are added as a reference (*(LF(no Mrg.))*) to show how the results are affected by the merge mechanism.

For *Setting 1* in Table 4.9, the delete threshold parameters are set according to *FA: Balanced* in Table 4.1 for *LF* ($T_{Delete} = 0.09$) and *GFnorm* ($T_{Delete} = 0.001$), but it is set to $T_{Delete} = 0.01$ for *GFdisc* so that all methods are provoked to delete some units. The results show that while *GFnorm* employs the strictest delete threshold parameter, it deletes more units than the other two methods. Since no forgetting occurs, this is mainly related to the difference in the normalization of the deletion indicator $\langle\!\langle 1 \rangle\!\rangle_i(t)$. I have discussed three different possible normalizations in Section 3.3.1, where the first one is similar to *GFnorm*'s delete, the second is applied to delete units with *GFdisc* and the third one is applied to *LF*. Especially, the first normalization approach is not considering that the life time of each unit is different in dynamic model selection approaches, resulting in units that are added

later in the run to be deleted more easily. One can observe that *LF* is performing best out of the three methods while having more unit manipulation occurrences than *GFdisc*. *GFnorm* performs worst of all three methods, likely because its delete mechanism affects the learning performance negatively. Overall, the experimental results show that the proposed changes work well in this testbed. The difference in learning performance is not only quite high but also statistically significant, which is supported by the p-values approximately equaling zero. These results emphasize the effectiveness of the localized deleting approach applied to *LF*. Furthermore, one can observe that the additional merge mechanism is able to further decrease the model complexity while having a similar learning performance for this test case when comparing the result of *LF(Prop.)* with the reference result *LF(no Mrg.)*.

For *Setting 2* in Table 4.9, the delete threshold parameters are set so that all three methods almost never delete units with $T_{Delete} = 0.01$ for *LF*, $T_{Delete} = 0.0001$ for *GFnorm* and $T_{Delete} = 0.001$ for *GFdisc*. This enables a comparison when the methods' learning behavior is not influenced by the deletion of units. Here, *GFnorm* and *GFdisc* perform slightly better than *LF*. Yet, this time the performance differences are relatively small compared with *Setting 1* and the p-values report only a medium significance of the differences. When no forgetting occurs, *GFdisc* and *LF* update methods reduce to the same function, so performance differences are the result of different unit manipulation mechanisms. Since deletion almost never occurs for all methods and the *GF* methods were able to perform better, the observed performance difference shows that the static threshold is able to affect the learning performance more positively than the proposed method. Considering the reference (*LF(no Mrg.)*), I conclude that the performance difference is mainly affected by the split mechanism with localized thresholding, while the merge mechanism has a positive effect on the learning performance. In this testbed, the data and noise distributions are balanced so that there is no apparent advantage of localizing the split decision. The merge mechanism helps to reduce the model complexity of *LF*, which is approximately 10% smaller than for the *GF* methods.

Finally, I compare the results of *LF* applying dynamic model selection with the results of *LF(Prop.)* using a static model complexity in Table 4.5 for the test cases where no forgetting occurs. Without model selection, *LF* has achieved a performance of $NMSE = 0.0267$ in the best case with a model complexity of 100 units. With dynamic model selection, *LF* performs better while employing a model complexity that is approximately only half the size for *Setting 1*. For *Setting 2*, the performance is worse than $NMSE = 0.0267$ but considering the smaller model complexity, this result is still reasonable and its likely that it would perform better when a higher

TABLE 4.10: Balanced Test with Forgetting

Discount	LF(Prop.)		GFnorm			GFdisc		
	NMSE	UnitNo.	NMSE	UnitNo.	p-Val	NMSE	UnitNo.	p-Val
b=30								
a=0.1	0.0184	47.98	**0.0059**	128	0	0.0075	135.12	0
a=0.01	**0.0143**	53.5	0.0508	109.96	0	0.0508	109.86	0
a=0.001	**0.0128**	57.56	0.1491	77.86	0	0.1575	78.16	0
b=150								
a=0.1	0.0189	45.86	**0.0062**	86.6	0	0.0087	96.22	0
a=0.01	**0.0178**	46.86	0.0242	128.04	0	0.0246	128.58	0
a=0.001	**0.0176**	47.26	0.0410	116.02	0	0.0403	116.4	0
b=1000								
a=0.1	0.0197	45.1	0.0165	36.7	0	**0.0159**	64.74	0
a=0.01	0.0194	45.08	**0.0120**	42.18	0	0.0137	70.56	0
a=0.001	0.0193	45.04	**0.0110**	46.32	0	0.0134	71.3	0

production of units would be allowed. These results also show that the selection of threshold parameters is important for the learning performance. Both settings in Table 4.9 are able to perform better than the static models with 25 units.

With Forgetting For the *balanced* testbed with forgetting, experimental results are presented in Table 4.10 with the best results for each discount schedules marked in bold. Here, *LF* is not the overall superior approach but shows the highest robustness over all discount schedules for both learning performance and network size. It also has the best learning performance on average over all 9 discount schedules. Additionally, it can be said that *LF* performs better than the *GF* methods when larger discounts are applied, because the learning behavior of the global forgetting methods becomes unstable in these cases. On the other hand, the two *GF* methods perform approximately the same on average and have shown similar performance behavior over all discount schedules. Yet, *GFnorm* is able to perform better than *GFdisc* in most cases. The two *GF* methods are able to perform best for discount schedules where their learning is stable. This applies for discount schedules that are near to one from the start ($b = 1000$) or reaching one very fast ($a = 0.1$). For higher discounts, the performance steadily decreases and observed over all discount schedules both *GF* methods exhibit a non robust behavior.

Again, it is interesting to observe the effects of the different delete mechanisms for the two *GF* methods. Although no delete manipulation counter data are presented, one can observe that *GFnorm* has a smaller network size than *GFdisc* for most discount schedules. This is likely related to a higher number of unit deletions. Also, the *GF* methods' network sizes become

TABLE 4.11: Imbalanced Test without Forgetting

Method	RMSE	Net. Size	Manipulation Counter				p-Val
			Prod.	Del.	Spl.	Mrg.	
Setting 1							
LF(Prop.)	**0.1044**	44.76	46.64	0.04	4.14	6.98	
(LF(no Mrg.))	(0.1011)	(49.8)	(45.04)	(0)	(3.76)	(/)	
GFnorm	0.1046	47.62	45.72	0	0.9	/	0.47
GFdisc	0.1046	47.62	45.72	0	0.9	/	0.47
Setting 2							
LF(Prop.)	**0.1047**	44.54	46.74	0.18	4.14	7.16	
(LF(no Mrg.))	(0.1011)	(49.8)	(45.04)	(0)	(3.76)	(/)	
GFnorm	0.6159	20.66	180.54	161.38	0.5	/	0.0
GFdisc	0.1210	45.34	48.48	4.94	0.8	/	0.0

similar for the discount schedules where *LF* performs best. The higher discounts prevent the normalization factors to grow larger than a certain value, which makes the normalization of both *GF* methods converge to the approximately same and relatively small value resulting in a similar deletion behavior. Although, the delete manipulation counter data for *LF* is also not presented, the deletion behavior was approximately the same for all discount schedules with only a slight increase for the larger discounts where $b = 30$, emphasizing the robustness of the deletion behavior.

Permutation tests were conducted comparing *LF* with either *GFnorm* or *GFdisc*. Regardless of whether the local or global forgetting method has performed better, the resulting p-values were approximately equal to zero in all cases proving the significance in the performance differences.

Furthermore, I compare the results of *LF* applying model selection with the results of *LF(Prop.)* without model selection in Table 4.5. Without model selection, *LF* has achieved learning performances between $NMSE = 0.0442$ and $NMSE = 0.0170$ for models with 25 units and between $NMSE = 0.0264$ and $NMSE = 0.0109$ for models with 100 units depending on the discount schedule. For *LF* with model selection, the performance is ranging between $NMSE = 0.0197$ and $NMSE = 0.0128$ for an average model complexity of approximately 50 units. So, while it not always performs better than the static model complexity with 100 units, it is able to achieve very good learning performance within a smaller performance range over all discount schedules with only half of the model complexity. This emphasizes the advantage of applying dynamic model selection to *LF*.

Imbalanced Testbed With Dynamic Model Selection

Without Forgetting For the *imbalanced* testbed without forgetting, *LF* and the *GF* methods are compared for discount $\lambda = 1$ and experimental results

are presented in Table 4.11, where the best results are marked in bold. Except for delete, manipulation parameters are set as stated in Table 4.1 for *FA: Imbalanced*. For delete, two threshold parameter settings are tested. In *Setting 1*, the thresholds are set to $T_{Delete} = 0.0001$ so that for all compared methods no units are deleted, while in *Setting 2* I have relaxed the thresholds to $T_{Delete} = 0.001$ to show the differences in learning performance and deletion behavior when deleting occurs. Again, I also added experimental results for *LF(Prop.)* without applying the merge mechanism (*LF(no Mrg.)*) as a reference for better interpretation.

For *Setting 1*, the performance difference is small and significance is low as indicated by the p-values. Here, *LF* performs slightly better than the two *GF* methods, and it possesses a lower model complexity. It can be remarked positively that *LF* performs better while the number of model manipulations is higher. Since this testbed has a highly imbalanced data distribution, there is a tendency that too many model manipulation occurrences negatively affect the learning performance of the NGnet. The NGnet units are dependent on each other due to the normalization and changes in model complexity come together with the need to adjust to the new model complexity. This is however difficult for units in regions that barely receive new data samples as it is the case in this testbed. Especially merge is then a problem. The frequently sampled region observes much more data, which provokes the development of more model redundancies leading to frequent merging. Yet, the relationship between all units has to be adjusted and learning performance then decreases for units in regions where not enough possibilities to adjust the relationship are available. The localized split threshold decision works however favorable for the imbalanced data distribution as it is apparent when comparing the reference result *LF(no Mrg.)* with *LF(Prop.)*. Even though the significance of the results is low, the results can still be interpreted positively in favor of the *LF* method.

In *Setting 2*, deletion occurs for all methods, although the number of deleted units is differing largely for each of them. Considering that the updates are the same for *LF* and *GFdisc* and the same delete thresholds are applied to all methods, the results give some insights about the different behavior of the three delete mechanisms. While the delete mechanism of *GFnorm* has a high increase in deletions, likely for the same reasons that were discussed previously for the balanced testbed and in Section 3.3.1, *GFdisc* and *LF* still exhibit a stable deletion behavior. Yet, *GFdisc* has already a higher number of deletions than *LF* and the difference between the two methods would become larger for bigger threshold parameters. Comparing the learning performances of the three methods for *Setting 2*, *LF* is able to perform best with high significance while possessing approximately the same model complexity as *GFdisc*. While *GFdisc*'s performance

TABLE 4.12: Imbalanced Test with Forgetting

Discount	LF (Prop.)		GFnorm			GFdisc		
	NMSE	UnitNo.	NMSE	UnitNo.	p-Val	NMSE	UnitNo.	p-Val
b=30								
a=0.1	**0.0999**	45.92	0.1411	107.48	0.00	0.1411	107.48	0.00
a=0.01	**0.1096**	46.9	0.5229	90.22	0.00	0.5230	90.28	0.00
a=0.001	**0.1110**	47.62	0.8960	49.68	0.00	0.9408	49.3	0.00
b=150								
a=0.1	**0.1039**	45.3	0.1368	100.82	0.00	0.1368	100.82	0.00
a=0.01	**0.1079**	44.84	0.3911	100.62	0.00	0.3868	100.4	0.00
a=0.001	**0.1107**	44.06	0.4555	98.12	0.00	0.4591	98.12	0.00
b=1000								
a=0.1	0.1049	44.34	**0.1042**	65.98	0.44	**0.1042**	65.98	0.44
a=0.01	**0.1048**	44.48	0.1321	78.8	0.00	0.1321	78.8	0.00
a=0.001	**0.1048**	44.48	0.1375	80.98	0.00	0.1375	80.98	0.00

decrease is already quite large considering the relatively small number of unit deletions, the performance of *GFnorm* decreases disastrously provoked by the bad deletion behavior. Overall, the results emphasize that my newly proposed manipulation mechanisms work well in the imbalanced environment, although the application of merge shows to have a slight negative effect on the learning performance.

With Forgetting For the *imbalanced* testbed with forgetting, the experimental results are presented in Table 4.12. Again, several discount schedules are applied and the best result for each schedule is marked in bold. Dynamic model manipulation parameters are set as stated in Table 4.1 for *FA: Imbalanced*. Because of the imbalanced data distribution and the applied discount, this testbed is more prone to negative interference.

The results show favorable performance for the proposed method *LF* in almost all cases with high significance proven by p-values equaling zero. There is one exception for the discount schedule ($a = 0.1$, $b = 1000$), where *LF* and the two *GF* methods have a similar learning performance with the *GF* methods performing slightly better on average. Yet, the high p-value shows that these results have statistically seen no significance. Also, it should be noted that *LF* is able to have a similar learning performance with a ca. 30% smaller model complexity. Overall, *LF* shows a high stability in learning performance and model complexities over the discount schedules and provides the best performance for the *imbalanced* testbed at $a = 0.1, b = 30$. On the other hand, the learning performances and model complexities of *GFnorm* and *GFdisc* are strongly varying and both perform very bad when the discount is high. These results emphasize the robustness of *LF* in negative interference prone testbeds and its advantage over the *GF*

TABLE 4.13: Results for Dynamic Testbeds

Discount/ Method	NMSE All	NMSE Last	UnitNo.	Manipulation Counter				p-Val
				Prod.	Del.	Spl.	Mrg.	
$\lambda = 1$								
LF(Prop.)	**0.0050**	**0.0085**	213.8	288.1	102.94	42.9	15.26	
GFnorm	0.3021	1.2828	112.26	23190.22	23079.38	0.42	/	0.0
GFdisc	0.0071	0.0095	219.6	218.58	0.4	0.42	/	0.0
$\lambda = 0.999$								
LF(Prop.)	**0.0055**	0.0078	206.42	294.2	112.82	41.16	17.12	
GFnorm	4.0411	**0.0058**	209.1	311.22	103.56	0.44	/	0.0
GFdisc	4.4606	**0.0058**	210.32	312.1	103.22	0.44	/	0.0

methods. *LF* is therefore preferable compared with the global forgetting methods.

Although, data about the manipulation occurrences are not presented to save place, I shortly discuss the deletion behavior of the three methods for this testbed. *LF* has similar model complexities for all discount schedules, which is also mirrored in a stable deletion behavior that does not differ much over all discounts. On the other hand, the two *GF* methods exhibit a less stable deletion behavior that is however similar for all discount schedules in this testbed. Interestingly, the deletion behavior shows a higher resemblance than for the *balanced* testbed, where big differences could be observed for $b = 1000$. This is probably because delete occurs in general more easily outside the frequently sampled regions so that the difference in the normalizers has less effect on the deletion behavior. One exception can be observed for the discount schedule $a = 0.001$ and $b = 30$, where both *GF* methods perform very bad, but there is a large difference with *GFdisc* performing even worse than *GFnorm*. There is however no apparent reason for this as the model complexity and the manipulation behavior are very similar. Maybe, the different timing of the model manipulations is somehow negatively affecting the performance of *GFdisc*. Overall, I conclude that both *GF* deletion mechanisms cannot behave well in this negative interference prone environment, showing that local learning strategies are necessary as employed by *LF*. These results emphasize the advantage of the local forgetting approach in online learning tasks where robustness is needed.

Dynamic Testbed with Dynamic Model Selection

Without Forgetting For the *dynamic* testbed without forgetting, the compared methods are trained with $\lambda(t) = 1$ over all time steps t. Here, the manipulation parameters are set as stated in Table 4.1 for *FA: Dynamic* except for *GFnorm* the delete threshold is set to an even smaller value with $T_{Delete} = 0.00001$. Fig. 4.4 illustrates for the three compared methods how

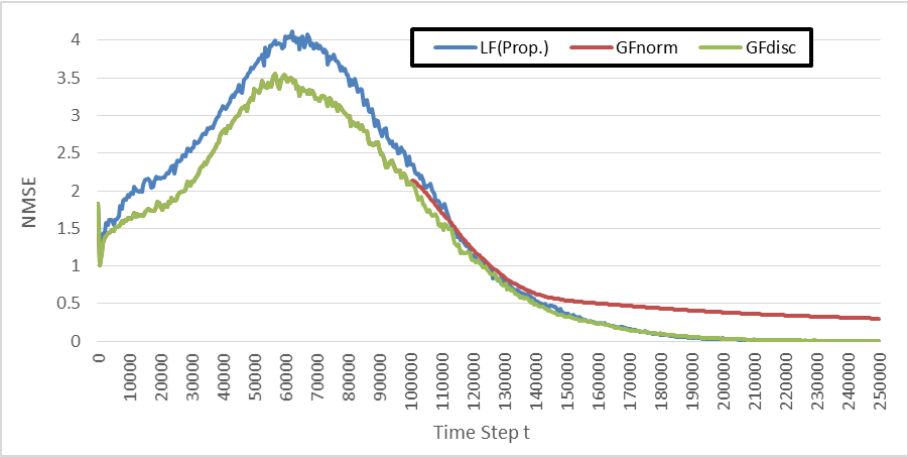


FIGURE 4.4: Running Error for Dynamic Testbed without Forgetting

the NGnet models' performance over the whole input distribution $[-1, 1]$ changes during training. *GFdisc* has a better learning performance over a large part of training, but eventually looses out to *LF* in the last quarter, which is however not visible in the figure because of the rough scale. *GFnorm* behaves similar to *GFdisc* for the first half of training, but then starts to perform worse than the other two methods, especially in the last third of training. The final performance results are presented in Table 4.13 for $\lambda = 1$, where one can observe that *LF* possesses better approximation capabilities not only over the whole input distribution (*NMSE All*) but also for the last interval $[0.2, 1]$ (*NMSE Last*). The favorable performance of *LF* is also supported by a statistical significance of the results with the p-value equaling zero.

The presented manipulation counter data in Table 4.13 give some insights about the dynamic model selection behavior of the three methods. For *GFnorm*, it is apparent why it performs worse than the other two methods, since it has a very high number of unit productions and deletions. Also, it is interesting to observe that *GFnorm* performs worse in the last input interval than over the whole input space. Without forgetting, the normalization factor $\eta(t)$ tends to grow smaller without limit and in the last interval *GFnorm* is then unable to keep newly added units. Each unit starts with a small weighted mean $\langle\!\langle 1 \rangle\!\rangle_i(t) = \eta(t)$ and is then prone to be deleted right away as $\eta(t)$ becomes very small. This assumption is also supported by the fact that *GFnorm* performs better over the whole input space, meaning that older units with higher weights $\langle\!\langle 1 \rangle\!\rangle_i(t)$ are not deleted. This deletion behavior demonstrates that the established NGnet training method *GFnorm* has difficulties with online learning tasks not only because of global forgetting but also because of the applied delete mechanisms. This

effect is drastically apparent for this testbed because the number of training data samples is large. For *LF* and *GFdisc*, the update reduces to the same function so that the differences in learning performance are only due to the different manipulation mechanisms. Here, *LF* is able to perform better with high significance and a slightly lower model complexity despite of the higher number of manipulations. Interestingly, a quite aggressive delete threshold has been selected for *LF* as the best performing one, while a very low threshold has been chosen for *GFdisc*. This shows the advantage of the localized delete mechanism. It was able to affect the learning performance positively while the global delete mechanism did not. Overall, the performance results show the superiority of the proposed method.

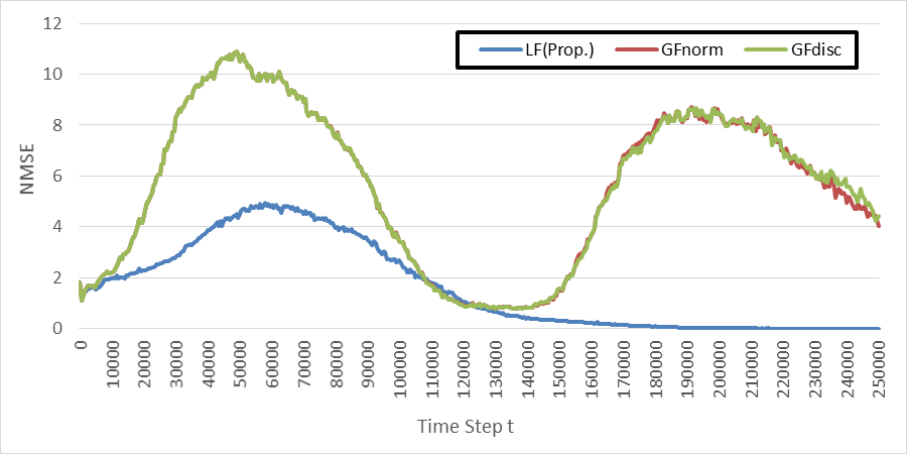


FIGURE 4.5: Running Error for Dynamic Testbed with Forgetting

With Forgetting In the *dynamic* testbed with forgetting, a static discount $\lambda(t) = 0.999$ is applied over the whole training time. This is equal to a discount schedule with $a = 0$ and $b = 1000$. I compare *LF* again with both *GFdisc* and *GFnorm* and apply the manipulation parameters as stated in Table 4.1 for *FA: Dynamic*.

In Fig. 4.5, a huge performance difference is visible for the running error of *LF* compared with *GFdisc* and *GFnorm*. At the second time quarter, the *GF* methods seem to be successfully learning and even perform better than *LF* for a short time frame. However, their performances then decrease and become almost as bad as at the begin of training for a second time. On the other hand, *LF* is able to steadily improve its approximation. The final performance results in Table 4.13 ($\lambda = 0.999$) also confirm that *LF* performs much better over the whole input space. Compared with the without forgetting case, this time the *GF* methods are able to perform better in the last

input interval *NMSE Last* but are unable to achieve an acceptable performance over the whole input space. The two *GF* methods perform similar to each other in regard to learning performance and dynamic model selection. Reasons for the bad performance of the *GF* methods are likely the global forgetting and the global deletion behavior. While *LF* is actually deleting more units because of the chosen delete threshold, it is still able to perform much better over the whole input space, which shows the importance of a local deletion approach in environments where the training data distribution is not i.i.d. All methods experience a decrease in performance compared with the non-forgetting test case, but only *LF* is able to perform similarly well. These results prove again the robustness of *LF* in regard to different learning situations.

Again, I shortly want to discuss the deletion behavior of the three methods. The delete thresholds of *LF* and *GFdisc* are equal to the ones in the *dynamic* without forgetting case, so it is interesting to compare the differences in deletion behavior. For *LF*, the number of unit deletions is only slightly increasing for the forgetting case, which confirms that the local deletion mechanism is behaving stable. On the other hand, while *GFdisc* has almost never deleted units before, the number of deletions increased when discount was applied. This shows that there is a high impact of global forgetting and the global time counter on the delete mechanism of *GFdisc*. In addition, one can observe that *GFnorm* has less deletions in this test case compared with the without forgetting case ($\lambda = 1$) while employing a larger delete threshold value. This highlights that *GFnorm* possesses a robuster deletion behavior when $\eta(t)$'s ceasing to zero is limited by the applied discount. I conclude that the delete mechanism of *GFnorm* is unsuited in general but especially when no forgetting occurs. Overall, it can be said that the proposed *LF* method is preferable for online learning tasks, especially when data distributions are not i.i.d.

4.3 Chaotic Time Series Approximation Tasks

In this section, I apply the NGnet to chaotic time series forecasting tasks that represent environments with non-i.i.d. data, since data samples are received in a time-dependent order. Two chaotic time series are tested: the Lorenz attractor and the Mackey-Glass chaotic time series. I have already observed for the Cross function approximation task in Section 4.2.2 that *GFnorm* has a higher potential of instable behavior than *GFdisc* and also discussed that this is likely related to its deletion mechanism. In test cases where *GFnorm* was able to perform stable, its performance has been generally similar to the one of *GFdisc*. Therefore, I only compare my proposed

learning method *LF* with *GFdisc* for the chaotic time series approximation tasks to save space.

4.3.1 Preparations

Some preparative decisions were taken before training and are introduced in the following.

Time Delay Embedding A time delay embedding method is commonly used for the approximation of chaotic time series. According to Takens' theorem (Takens 1985), it is possible to reconstruct the chaotic dynamics from one dimension, e.g. dimension x. A next state $x(t+P)$ is then defined by the following delay coordinate

$$x(t+P) = \{x(t), x(t-\Delta t), \ldots, x(t-(d-1)\Delta t)\}, \tag{4.10}$$

where d is the embedding dimension and Δt is the embedding delay. The reconstruction of the chaotic dynamics is dependent on an appropriate choice of d and Δt, and therefore the parameters are chosen separately for the two learning tasks. P is the number of prediction steps ahead.

Performance Evaluation For the chaotic time series approximation tasks, I evaluate the learning performance of the tested methods with the Root Mean Square Error (RMSE). The RMSE is defined by

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(y(t) - \hat{f}(x(t)))^2}{T}}, \tag{4.11}$$

where T is here the number of test data samples. The RMSE results represent then the short-term generalization performance of the trained NGnets for one-step ahead predictions.

4.3.2 Lorenz Attractor

For the first chaotic time series approximation task, the NGnet is applied to the Lorenz attractor (Lorenz 1963) that is defined by

$$\begin{aligned} \dot{x} &= a(y-x), \\ \dot{y} &= bx - y - xz, \\ \dot{z} &= xy - cz. \end{aligned} \tag{4.12}$$

For the commonly used parameters $a = 10, b = 28, c = 8/3$, the attractor possesses then chaotic behavior. Attractor trajectories are obtained by fourth order Runge-Kutta integration with an integration time step $t_i =$

0.001. The vector notation of the attractor is $\dot{X} = F(X)$ where $X \equiv (x, y, z)'$ and F denotes the vector field. Additionally, some noise can be added to each observation with

$$Y(t) = X(t) + \xi(t), \tag{4.13}$$

where the noise $\xi(t)$ is a white Gaussian noise with zero mean and some standard deviation SD.

Preparations

Further preparative steps are necessary to apply the NGnet to this learning task. Dynamic model selection is applied to the compared methods and threshold parameters are selected according to the values stated for *Lorenz Attractor* in Table 4.1.

Time Delay Parameters Appropriate parameters have to be chosen for the time delay embedding method. For the Lorenz attractor, $\Delta t = 0.15$ and $d = 3$ are sufficient to reconstruct the chaotic dynamics (Ishii and Sato 2001). The number of prediction steps ahead P is set to $P = 1$.

Training and Test Data The NGnets are trained with 10,000 training data samples sequentially obtained from one trajectory of the attractor with a sample time step $t_s = 0.01$. Short-term prediction accuracy is evaluated with two test sets: one is noiseless (*RMSE1*) and another one added white Gaussian noise with $SD = 0.2$ (*RMSE2*). For both test sets, 10,000 test data samples are randomly obtained from several different trajectories.

Correlation Dimension (CD) Additionally to the short-term performance, it is possible to evaluate the long-term characteristics of the trained NGnet models for the Lorenz attractor (Cowper *et al.* 2002). For long-term predictions, recursively predicted trajectories are obtained, where recursive means that the NGnets use their own predicted outputs as the input for the next prediction step. Because of the chaotic dynamics, the recursively obtained trajectory starts to differ largely from the original trajectory even when obtained from the same initial state. It is therefore not possible to evaluate the long-term characteristics by direct comparison.

The correlation dimension (CD) (Grassberger and Procaccia 1983) measures properties of a chaotic time serie's trajectory and can be used to evaluate the long-term characteristics of the trained NGnet models. For each trained model, three recursively predicted trajectories are obtaiend with 30,000 samples and different initial states. Then, CDs are calculated for each trajectory and their average is presented in the test results. The CD

TABLE 4.14: Lorenz Attractor Testbed Results with b=150

Discount/ Data Noise	LF (Prop.)				GFdisc			
	RMSE1	RMSE2	Size	CD	RMSE1	RMSE2	Size	CD
a=0.1								
SD=0.0	**0.1527**	**0.4843**	172	**1.87**	0.1805	0.7330	435	1.61
SD=0.1	**0.1757**	**0.4632**	169	**1.84**	0.1875	0.4961	399	1.82
SD=0.2	**0.2012**	**0.4538**	174	**1.78**	0.2921	0.5041	453	1.69
SD=0.3	**0.2430**	**0.4626**	207	1.76	0.3579	0.5428	584	**1.79**
a=0.01								
SD=0.0	**0.1506**	**0.4897**	173	**1.86**	0.6761	1.0421	494	0.32
SD=0.1	**0.1742**	**0.4645**	169	**1.83**	0.6504	0.9003	458	0.00
SD=0.2	**0.2006**	**0.4540**	175	**1.79**	0.5669	0.7459	511	1.49
SD=0.3	**0.2407**	**0.4622**	211	**1.78**	0.6951	0.8234	555	1.43
a=0.001								
SD=0.0	**0.1499**	**0.4912**	173	**1.87**	0.8621	1.1928	434	1.05
SD=0.1	**0.1737**	**0.4650**	169	**1.84**	0.7642	1.0197	412	1.09
SD=0.2	**0.2003**	**0.4539**	175	**1.78**	0.8382	0.9852	463	1.03
SD=0.3	**0.2404**	**0.4621**	211	**1.77**	0.9873	1.0708	519	1.58

of a real Lorenz attractor is $CD_{Original} = 2.05 \pm 0.01$ (Grassberger and Procaccia 1983), while it is approximately $CD_{Embedded} = 2.021$ for a trajectory obtained by the embedding delay method (Ishii and Sato 2001). The nearer a calculated CD is to the real CD the better are the long-term characteristics of the trained model.

Experimental Results

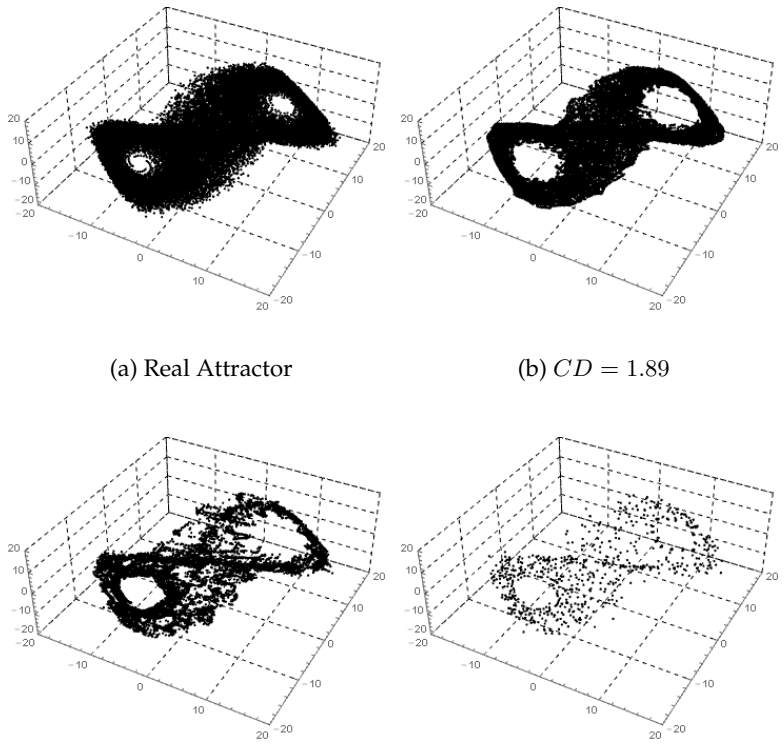
The effectiveness of the proposed method *LF* is compared with *GFdisc* for several discount schedules with initial settings $b = 150$ and $b = 1000$ in Table 4.14 and Table 4.15 respectively. For each discount schedule, I present results obtained from training runs with different grades of noise added to the data, denoted by the noise standard deviation SD. The results show that *LF* has a robust short-term performance in comparison with *GFdisc*, since it is able to perform well for all test cases and the full range of the achieved learning performances is much smaller. The long-term characteristics of *LF* are also robust, since there is no CD lower than 1.76 and CDs are similar within each training data noise level irrespective of the applied discount schedule. On the other hand, *GFdisc* is not able to perform robustly over the different discount schedules. Short-term performance is decreasing a lot when higher discounts are applied and it performs especially bad over all noise levels for two out of three discount schedules with $b = 150$. There are a few cases where *GFdisc* has either better short-term performance for

TABLE 4.15: Lorenz Attractor Testbed Results with b=1000

Discount/ Data Noise	LF (Prop.)				GFdisc			
	RMSE1	RMSE2	Size	CD	RMSE1	RMSE2	Size	CD
a=0.1								
SD=0.0	0.2329	**0.4826**	172	**1.86**	**0.1259**	0.7070	200	0.00
SD=0.1	0.2140	**0.4626**	169	1.83	**0.1537**	0.5009	181	**1.89**
SD=0.2	**0.2060**	**0.4539**	174	1.78	0.2218	0.4609	211	**1.81**
SD=0.3	**0.2143**	**0.4629**	206	**1.78**	0.2481	0.4744	263	1.72
a=0.01								
SD=0.0	**0.1534**	**0.4829**	172	**1.86**	0.1666	0.8423	225	0.36
SD=0.1	0.1764	**0.4627**	169	**1.83**	**0.1716**	0.5202	198	1.82
SD=0.2	**0.2016**	**0.4539**	174	**1.79**	0.2456	0.4887	246	1.73
SD=0.3	**0.2434**	**0.4629**	206	**1.77**	0.3107	0.5153	314	1.71
a=0.001								
SD=0.0	**0.1534**	**0.4830**	172	**1.85**	0.1735	0.8198	226	0.23
SD=0.1	**0.1764**	**0.4627**	169	1.82	0.1773	0.5166	203	**1.84**
SD=0.2	**0.2016**	**0.4539**	174	**1.79**	0.2537	0.4923	252	1.73
SD=0.3	**0.2434**	**0.4629**	206	**1.77**	0.3180	0.5175	321	1.74

the *RMSE1* test data or better long-term characteristics than *LF*, when discounts are small and mainly for $b = 1000$, but overall it misses the robustness shown by the proposed *LF* method. In addition, *GFdisc* has also more difficulties to exhibit good performance for the *RMSE2* test data set, where *LF* performs better regardless of the discount schedule. The results in Table 4.14 and Table 4.15 also show that better short-term performance is not necessarily an indicator for good long-term characteristics. For some cases, *LF* has better long-term characteristics even when short-term performance is worse than *GFdisc*'s for *RMSE1*. This phenomenon has also been noted by Cowper *et al.* (2002). Similarly to the function approximation experiments, one can observe again that *GFdisc* has difficulties to show a robust learning performance while *LF* is able to perform robustly over all testbeds. The same is true for the long-term characteristics, where *GFdisc* achieves only five CD values over 1.8 and one additional CD that is larger than 1.75. On the other hand, *LF* has achieved 12 CD values over 1.8 and all other CDs are at least higher than 1.75. Overall, it can be concluded that *LF* is performing clearly better than *GFdisc*.

In Fig. 4.6(a) - 4.6(d), some examples are illustrated of how different CD values relate to different representations of the long-term characteristics. 30,000 data points are presented in all four figures, but Fig. 4.6(d) seems much sparser than the rest because the recursive prediction always returns to the same points. Fig. 4.6(c) looks already more like the real attractor in



(a) Real Attractor (b) $CD = 1.89$

(c) $CD = 1.09$ (d) $CD = 0.23$

FIGURE 4.6: Real attractor (a) compared with recursive predictions (b) - (d)

Fig. 4.6(a) but still has some holes. Fig. 4.6(b) has a compact point distribution in space similar to the real attractor. All learned models have difficulties to represent the inner edges accurately resulting in the difference to the real attractor's CD.

4.3.3 Mackey-Glass Chaotic Time Series

The Mackey-Glass (MG) chaotic time series (Mackey and Glass 1977) is another popular example of a chaotic dynamic time series approximation task. It is also often referenced in the literature for testing neural network learning approaches, which makes it a good testbed to compare the proposed method not only with *GFdisc* but also with other self-constructing online learning approaches. The Mackey-Glass chaotic time series is defined as

$$x(t+1) = (1-a)x(t) + \frac{bx(t-\tau)}{1+x^{10}(t-\tau)}, \tag{4.14}$$

where the parameters are normally chosen to be $a = 0.1$, $b = 0.2$ and $\tau = 17$. The prediction problem is again described as a time delay embedding problem, where the future value $x(t+P)$ is predicted from past values embedded in a delay coordinate as in Eq. (4.10). The embedding delay

TABLE 4.16: MG 6 steps ahead prediction accuracy

Network	UnitNo.	RMSE	
		Training	Test
RBF-AFS	21	0.0107	0.0128
OLS	13	0.158	0.0162
DFNN	10	0.0082	**0.0083**
FAOS-PFNN	11	0.0073	0.127
GEBF-OSFNN	10	0.0091	0.0087
LF(Prop.)	33	**0.0056**	0.0086
GFdisc	38	0.0083	0.0098

parameter is set to $\Delta t = 6$ and the embedding dimension is set to $d = 4$ for the experiments.

For this learning task, *LF* is compared with *GFdisc* in two testbeds, again applying dynamic model selection to both methods. The selected threshold parameters are stated in Table 4.1 separately for both testbeds marked with a beginning *MG Time Series*. Here, it was necessary to choose a very high produce parameter ($T_{Produce} = 500.0$) to provoke early unit additions. The produce threshold parameter is dependent on the calculated probability density function over the input and output space, so the appropriate threshold value is highly dependent on the characteristics of the underlying testbed. In case of the MG time series, it is possible to use smaller produce thresholds, e.g. $T_{Produce} = 10.0$, but this results in new units being added more slowly and then they have less time to approximate the underlying functional relationship of the learning problem. This is especially a problem here because the number of available training data is strongly limited and the training data is applied only once for learning. Two different MG testbeds are applied with 6-step (Section 4.3.3) and 50-step (Section 4.3.3) ahead prediction, where both have a limited number of training data with 1,000 and 4,000 samples respectively.

First Testbed: 6-step ahead prediction

In the first testbed, the NGnets are trained for a MG time series with a 6-step ahead prediction ($P = 6$). Here, the initial condition of the model is set to $x(0) = 1.2$. These settings are chosen in reference to a paper by Wang (2011) to enable a direct comparison with the tested methods in the paper. Similarly to the related work, I only evaluate the short-term prediction accuracy for training and test data. The training and test data are extracted from the same time series trajectory, starting at sampling time step $t = 124$. The first 1,000 samples are obtained for the training data and the second 1,000 samples (from $t = 1124$) are obtained for the test data set.

In Table 4.16, the obtained performance results are presented for *LF* and *GFdisc*, which are also compared with several other constructive learning approaches where performance results are taken from Wang (2011). The referenced paper has proposed a generalized ellipsoidal basis function based online self-constructing fuzzy neural network (GEBF-OSFNN). Most of the other compared methods are also based on fuzzy neural network architectures: RBF based adaptive fuzzy system (RBF-AFS), dynamic fuzzy neural network (DFNN) and fast and accurate online self-organizing scheme for parsimonious fuzzy neural networks (FAOS-PFNN). Except the NGnets learning approaches, only one of the compared method is not fuzzy in Table 4.16: the orthogonal least squares algorithm for the RBF network (OLS). OLS is constructive but not applicable incrementally as the whole batch of training data is necessary for the construction. I also have shortly mentioned the OLS method in Section 2.2.3. For both NGnet approaches, I present the best performing results in the table with discount schedules set to $(a = 0.001, b = 5)$ for *LF* and $(a = 0.001, b = 1000)$ for *GFdisc*.

The results in Table 4.16 show that the NGnet with *LF* is able to perform very well compared with the other methods. It is also able to outperform *GFdisc* in regard to both training and test data prediction. For the two NGnet methods, the model complexity is higher than for the fuzzy neural networks and also than for the OLS method, where the OLS method is however not able to perform very well. The fuzzy networks have a number of units that is equal to the number of fuzzy rules. In my opinion, the higher model complexity is a natural result of the different network architectures where the NGnet units are more simple and can represent less information than each of the fuzzy rules. In addition, the increased model complexity is not really a problem for the proposed *LF*, because only a limited number of units is updated at every time step reducing its computational heaviness.

Second Testbed: 50-step ahead prediction

For the second testbed, the number of prediction steps ahead is set to $P = 50$ and the initial condition of the model to $x(0) = 0.3$. 4,000 training data samples are obtained starting from the sampling time step $t = 1$ and 500 test data samples are obtained from $t = 4001$. I refer to Liang *et al.* (2006) for the comparison of the proposed method with other self-constructive online approaches, including RAN (Platt 1991), RAN-EKF (Kadirkamanathan and Niranjan 1993), MRAN (Lu *et al.* 1997), GGAP-RBF (Huang *et al.* 2005) and OS-ELM (Liang *et al.* 2006). The first four approaches are based on RBF networks and also have been discussed shortly in Section 2.2.3, while the last approach is based on the ideas of extreme learning machines (ELM).

The results in Table 4.17 show that the proposed method is able to outperform all compared methods. For both NGnet learning strategies, I present

TABLE 4.17: MG 50 steps ahead prediction accuracy

Network	UnitNo.	RMSE	
		Training	Test
RAN	39	0.1006	0.0466
RAN-EKF	23	0.0726	0.0240
MRAN	16	0.1101	0.0337
GGAP-RBF	13	0.0700	0.0368
GEBF-OSFNN	12	0.0316	0.0218
OS-ELM (RBF)	120	0.0184	0.0186
OS-ELM (Sigmoid)	120	0.0177	0.0183
LF(Prop.)	45	**0.0137**	**0.0140**
GFdisc	45	0.0165	0.0141

the best results, where the discount schedules are set to $(a = 0.001, b = 25)$ for *LF* and $(a = 0.001, b = 1000)$ for *GFdisc*. *GFdisc* is also able to perform well coming in second place in comparison with all methods. Similar to the first MG testbed, the performance gap between *GFdisc* and *LF* is quite large for the training data set. This leads to the assumption that a *GF* method has more difficulties to perform well on previously learned data probably due to the forgetting. Both NGnets have the same network model complexity, which is approximately a third of the network size of the OS-ELM, while performing better than this method. All other compared methods have smaller network sizes but also perform much worse. I assume that likely a certain network complexity is necessary to perform well on this learning task.

4.4 Reinforcement Learning Task

For the last experiment, I apply the NGnet to a reinforcement learning (RL) task. RL provides an environment that is naturally prone to negative interference as received training data samples are not i.i.d. While a learning agent explores the environment to find a good policy for the learning task, it will visit some regions of the input space more frequently than other regions. This is especially the case after learning converges to a policy and exploration of other regions becomes less frequent. RL makes therefore an interesting testbed to evaluate the effectiveness of the proposed *LF* method in regard to *GFdisc*.

The NGnet is applied to a classical RL benchmark problem with a continuous state and action space that considers the control of a simple inverted pendulum with limited torque (Doya 2000). The learning goal is to swing up the pendulum and fix it in the upright position indefinitely, see also Fig. 4.7. The problem is described by a 2-dimensional state space that

TABLE 4.18: Physical Parameters for Simple Pendulum Dynamics

Parameter	Description	Value
g	gravity (m/sec^2)	9.81
l	pendulum length (m)	1.0
m_p	pendulum mass (kg)	1.0
μ_p	torque	0.01
F	force (N)	{5,0,-5}

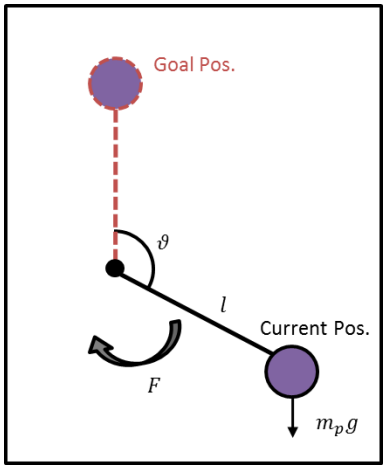


FIGURE 4.7: Control of a inverted pendulum with limited torque (inspired by Doya (2000))

consists of the angular position of the pendulum ϑ, measured in regard to the upright position, and its angular velocity $\dot{\vartheta}$. Here, ϑ takes values in the interval $[-\pi, \pi]$ and $\dot{\vartheta}$ is limited to the value range $[-8, 8]$. The dynamics of the pendulum are defined by the following differential equation

$$\ddot{\vartheta} = \frac{(-\mu_p \dot{\vartheta} + m_p g l \sin(\vartheta) + F)}{(m_p l^2)}. \tag{4.15}$$

The equation includes several physical parameters that are explained in Table 4.18. A numerical solution is obtained for the differential equation with the fourth order Runge-Kutta method, where I use a sampling time step $\tau = 0.1$. The experiment is then conducted in episodes of 70 time steps, which is equal to 7 seconds of simulated time. Although the action space is continuous, I have discretized it to three actions, where either positive force, negative force or no force is applied to the pendulum. The applicable force is selected so that it is smaller than a maximal load torque $m_p g l$ and the pendulum needs to be swung several times to build up enough kinetic energy to reach the upright from the downward position (Doya 2000).

4.4.1 Preparations

Some preparative steps are taken before learning that are explained in the following.

RL General Settings For the RL task, I apply a greedy exploration strategy where the learning agent always chooses the action $a_t = \pi(s_t) = \arg\max_a Q^\pi(s_t, a)$. In addition, the reward is obtained by the negative of the absolute value of the distance to the goal position, where the goal is $\vartheta = 0$ and the reward function is then equal to $r(s, a) = -|\vartheta|$ (Agostini and Celaya 2016).

Training and Test Runs The NGnet has to approximate the Q-function with input samples $x = (s_t, a_t)$, described by the current state $s_t = (\vartheta, \dot{\vartheta})$ and the selected action a_t, and target outputs that are calculated according to Eq. (2.4). The discount γ in Eq. (2.4) is fixed at $\gamma = 0.85$. For all experiments, I have conducted 20 independent runs with 1000 episodes each and only averages are presented as results. Each episode consists of a training run and a test run with 7 seconds of simulation time each. While the training run is mainly for learning a better policy, the test runs are conducted to evaluate the current policy when no learning takes place.

Pendulum Initialization The angular velocity $\dot{\vartheta}$ is always initialized to zero for both training and test runs. The pendulum's initial position is always set to the downward position for the test runs with $\vartheta = \pi$, but for the training runs ϑ is initialized differently in two considered testbeds. In the first testbed, ϑ is initialized randomly over the full value range $[-\pi, \pi]$. For the second testbed, it is set to $\vartheta = \pi$.

Learning Methods and Evaluation Similarly to the Lorenz attractor learning task, I only compare the proposed *LF* method with *GFdisc* for the pendulum RL task. The applied manipulation threshold parameters are stated for *Reinforcement Learning* in Table 4.1 and learning performance is evaluated by the accumulated amount of reward in each episode.

4.4.2 First Testbed

For the first testbed, the NGnet has to learn a good swing-up policy for a pendulum that can take any possible initial position int its state space. Therefore, the level of exploration is high and some hints of the goal are given when the initial pendulum position is near to the upright position for a training run. On the other hand, the learning performance is evaluated only for one initial position in the test runs where $\vartheta = \pi$. Therefore, learning is expected to improve rather slowly. First, the two forgetting methods are

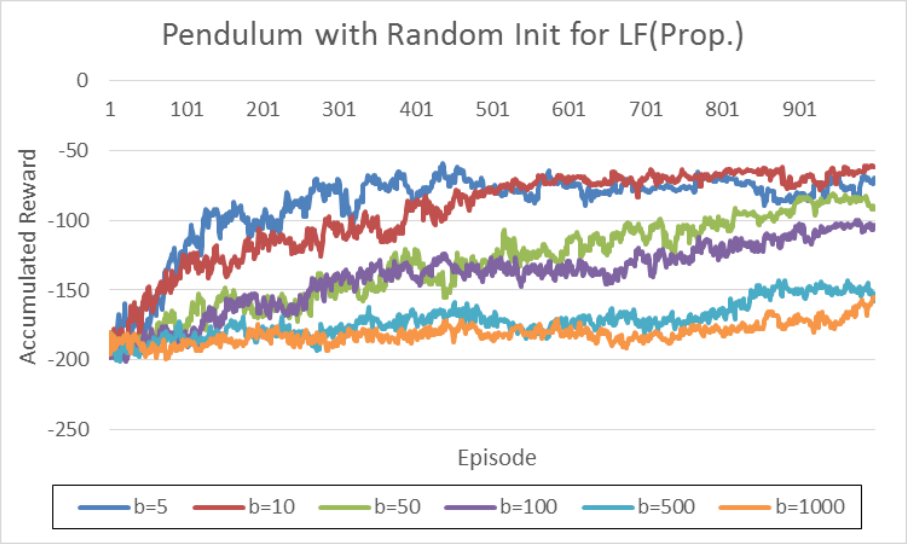


FIGURE 4.8: First Testbed with $a = 0.001$ and Different b for *LF(Prop.)*

evaluated separately for different discount schedules, where $a = 0.001$ and several initial discount values b are tested. Then, I will compare the best performing runs for each method with each other.

For *LF(Prop.)*, the trend of the learning performance is presented in Fig. 4.8, which shows how the learning performance changes over the episodes for 6 different initial discount values. The discounts represent a broad range of initial values that go from very large discounts ($b = 5$) to very small ones ($b = 1000$). It is apparent that the trend of the learning performance is highly dependent on the applied discount and a high discount is necessary to achieve an increase in learning performance for this task. Even over the long run of 1000 episodes, small discounts, $b = \{500, 1000\}$, are not able to show an improving trend in the learning performance. Also, one can observe that a higher initial discount leads to a faster improvement in learning performance, while initial discounts $b \geq 50$ take off to a slow start and it takes a long time before these test cases start to actually improve on the task. The best performing test cases have a very high initial discount with $b = 5$ or $b = 10$. Here, one can observe that the highest discount ($b = 5$) is improving faster, but in the long run the high discount has a slight negative effect on the learning performance as some instabilities are observed for the last learning phase. The more modest discount $b = 10$ performs then better in the last learning phase. Similarly, it was possible to observe better performance for the *LF* method when the discount is higher for the function and chaotic time series approximation tasks, but the results here further emphasize the importance of a high discount for learning success. The improvements of *LF* are in general slow if the applied discount is small.

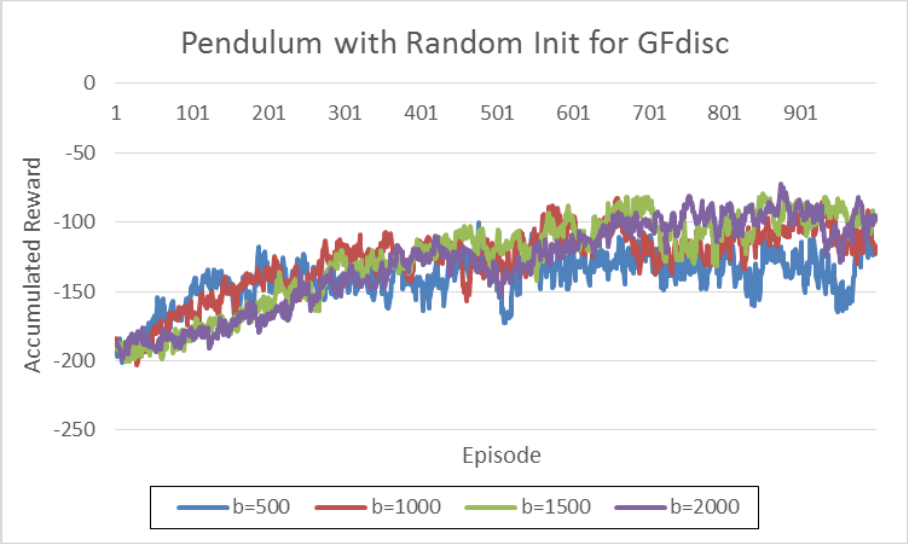


FIGURE 4.9: First Testbed with $a = 0.001$ and Different b for *GFdisc*

On the other hand, while high discounts can help to deal with this problem, one has to be careful that the high discount does not start to affect the learning performance negatively in a later phase of learning. This problem likely depends a lot on the length of the learning phase.

In Fig. 4.9, the trend of the *GFdisc* learning performance is presented over the whole training time. Again, several initial discount values are tested, where however the range of discounts is set from small ($b = 500$) to very small ($b = 2000$). I only choose relatively small discount for the comparison, since *GFdisc* behavior becomes instable when higher discounts are applied. This can already be observed for a discount with initial $b = 500$, which performs best of all discount schedules in the first phase of learning but then deteriorates and finally performs worst in the last learning phase. It also has the highest fluctuation of accumulated rewards between episodes. Furthermore, if one compares the fluctuations of all discounts with the ones of *LF* in Fig. 4.8, then it is noticeable that the fluctuations are overall much higher for *GFdisc*. This is again related to the negative effect of the global approach and emphasizes the higher robustness of a localized approach. Overall, the performance differences are not as visible as for *LF*, which is probably caused by the smaller difference in the applied discount schedules. For all applied discounts, *GFdisc* has difficulties to converge to a good action policy.

Finally, I want to compare the two best performing results for *LF* and *GFdisc* with each other. The trends of the best performing discount schedules are presented in Fig. 4.10, where I have selected an initial discount value $b = 10$ for *LF* and $b = 2000$ for *GFdisc*. The difference in the trends

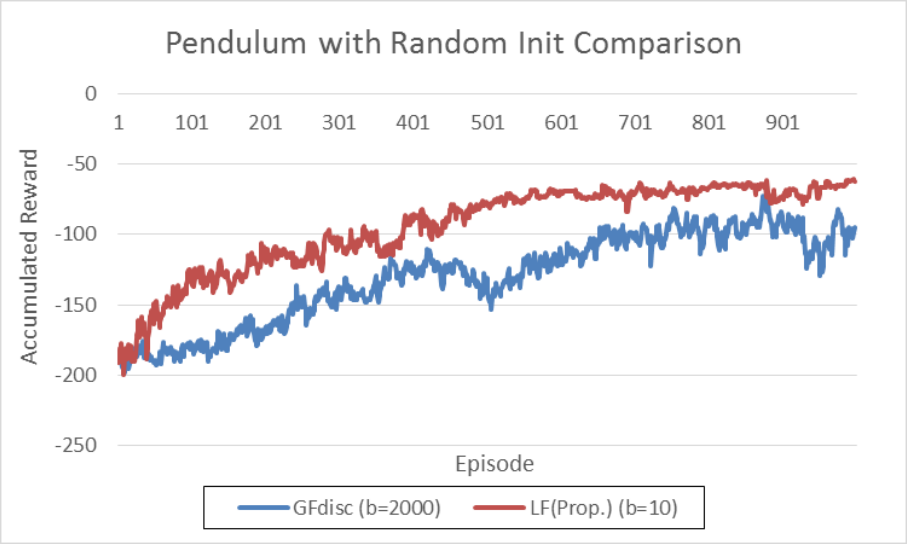


FIGURE 4.10: Compare Best Performance for First Testbed

then clearly shows the better performance of *LF*, which is not only able to perform better over all learning episodes but also converge to a much more stable solution with less fluctuations in the accumulated reward.

4.4.3 Second Testbed

For the second testbed, the angular position of the pendulum is initialized always in the same position, $\vartheta = \pi$, for all training and test runs. The grade of exploration is therefore low and it is only necessary to learn the trajectory for one initial state. For this testbed, I only present the best results for both methods, since the overall trend of different discount schedules is similar to the first testbed. The learning performance of *LF* and *GFdisc* is compared in Fig. 4.11, where the initial discount value is set to $b = 5$ for *LF* and $b = 1000$ for *GFdisc*. Interestingly, both methods apply a higher initial discount for this testbed, which is probably related to the static initial state of the pendulum. Here, the learning agent does not explore different initial pendulum states but always starts in the same position. This also means that more data samples are observed in the same regions of the input space, and this testbed has a less imbalanced data distribution than the first one. Therefore, it is possible to apply higher discounts for both testbeds without experiencing a deterioration of performance in the later learning phases.

The trend of the learning performances can be observed in Fig. 4.11. Since there is no random exploration of different initial states, all random factors are eliminated and the NGnet training is discrete. In other words, all test runs achieve the same learning performance as long as some of the NGnet parameter settings are not changed. Therefore, the presented results

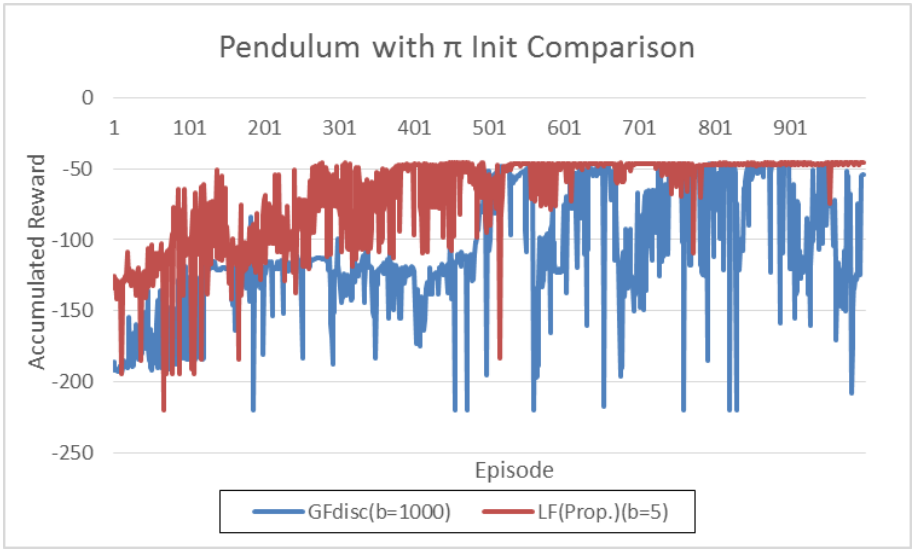


FIGURE 4.11: Compare Best Performance for Second Testbed

look very different from the first testbed, here the accumulated reward fluctuation is much higher for both methods as there is no soothing effect by averaging the results. An accumulated reward of -50 is the approximately best that can be achieved, since the pendulum needs some time to swing up to the upright position. Both methods are able to achieve high rewards for this testbed, where even *GFdisc* is able to reach a near to optimal performance from time to time. For the first testbed, the average performance is not able to converge to an optimum for both methods, probably because of the high level of exploration that can negatively affect the learning. Then, it prevents the convergence of the trained model to a good action policy for some of the test runs, resulting in a non-optimal performance on average. For this testbed, the fluctuation for *GFdisc* is high even in the last phase of learning, but *LF* is able to achieve almost constantly high values near to optimum after the 400th learning episode. The stability of *LF* performance increases further with advancement of the training time. This shows that *LF* is able to learn a good action trajectory for the initial pendulum position $\vartheta = \pi$, which is the ultimate goal of learning evaluated in the test episodes. Overall, *LF* is able to achieve higher rewards much earlier and much more steady than *GFdisc*, emphasizing again its preferable robustness in negative interference prone environments.

Chapter 5

Discussion

In this chapter, I discuss properties of the proposed method in regard to the results obtained for the conducted experiments in Chapter 4. The discussion is largely divided into two parts: properties of the localized forgetting method in Section 5.1 and properties of the dynamic model selection mechanisms in Section 5.2.

5.1 Updates with Localized Forgetting

In this section, I focus on the localized forgetting update method of the NGnet and discuss the properties of my proposed approach in comparison with the previous local forgetting method (Section 5.1.1) and the global forgetting method (Section 5.1.2).

5.1.1 Comparison with the Previous Local Forgetting Method

I have reconsidered the derivation of the localized forgetting approach in Section 3.1 to eliminate the dependency of the update weight on discount factor $\lambda(t)$. As a result, I have derived a new localized forgetting update method that applies the discount only for forgetting and becomes applicable over the whole numerical range of discount factor $\lambda(t)$. In Section 4.2.2, some experiments are conducted for a function approximation task to compare the previous LF method with my newly derived one. Overall, the experimental results have shown only a very small difference in learning performance when forgetting occurred, and the difference decreases further with smaller discounts. On the other hand, the new *LF(Prop.)* method is apparently better than the old *LF(Prev.)* when no discount ($\lambda(t) = 1$) is applied over all t, because *LF(Prev.)* is then unable to update its network parameters. Therefore, the main aim of my new proposal has been fulfilled.

Additionally, I want to discuss shortly the reasons for the less apparent differences in the learning performance of the two LF methods when forgetting occurs (Backhus *et al.* 2016a). Any numerical difference in the performance of the two approaches is affected only by the update factors. The old update factor $\Omega(P_i(t)) = \frac{1-\lambda(t)^{P_i(t)}}{1-\lambda(t)}$ depends on the current discount

TABLE 5.1: Sample Values for *LF(Prev.)* Update Factor

Discount $\lambda(t)$	Posterior Probability $P_i(t)$				
	0.01	0.2	0.4	0.6	0.8
0.9	0.01053	0.20852	0.41268	0.61260	0.80834
0.99	0.01005	0.20080	0.40121	0.60121	0.80080
0.999	0.01000	0.20008	0.40012	0.60012	0.80008

factor $\lambda(t)$ and unit weight $P_i(t)$, while the newly proposed update factor is only dependent on and equal to $P_i(t)$. Therefore, the discount $\lambda(t)$ plays an important role in the numerical differences between both methods. In Table 5.1, some sample values are presented for the update factor of *LF(Prev.)* that show how different values of $P_i(t)$ and $\lambda(t)$ influence it. The difference becomes larger as further $\lambda(t)$ is away from one and $P_i(t)$ is near to 0.5. For the examples in the table, the difference between $P_i(t)$ and $\Omega(P_i(t)) = \frac{1-\lambda(t)^{P_i(t)}}{1-\lambda(t)}$ is then largest for discount $\lambda(t) = 0.9$, which is equal to a discount schedule with $a = 0.0$ and $b = 10$. I have conducted experiments with this and other discount schedules and the results are presented in Table 4.5 of Section 4.2.2. The experimental results show similar tendencies as discussed here with higher performance differences when the applied discount is larger. Overall, I conclude that no large performance differences can be expected since the *LF(Prev.)* update factor's is still similar to the *LF(Prop.)* update factor due to its characteristics, and the influence on the learning performance is therefore limited except for discount $\lambda(t) = 1$.

5.1.2 Comparison of Local and Global Forgetting

For most of the conducted experiments, I have compared the newly proposed *LF* method with two *GF* methods and all methods have employed some dynamic model selection mechanisms. In this Section, I attempt to summarize the differences in learning performance that were mainly caused by the applied forgetting approaches by discussing benefits and limitations of LF.

Benefits

Robustness is one of the biggest advantages of the LF update method compared with GF. This was observable throughout the experiments in Chapter 4 in two manners. First, the LF updates have shown a higher robustness against the application of different discount schedules. While LF is not performing equally well for all tested cases, it is able to perform with a certain grade of robustness for different discounts. On the other hand, GF has difficulties to perform well when the applied discounts are too large

resulting even in a catastrophic learning behavior. Because large discounts make a GF network forget learned information much faster than new information can be learned, the units become smaller and smaller and eventually the network output becomes *not a number* in cases where no unit manipulation is applied and no other prearrangements are taken. This problem can be avoided with dynamic unit manipulation, where units are deleted before they become too small. The same problem translates also to the second perception on robustness, where LF is able to deal much better with non i.i.d. data than GF. Again, this is related to units globally forgetting information and becoming eventually unable to present information in regions that are sampled sparsely or only at an earlier point of learning.

Computational Complexity can be decreased when applying localized forgetting, because the properties of LF make it possible to reduce the number of updated units (see discussion in Section 3.2). This is a big advantage compared with GF, where the computational cost of updates grows linearly with the model complexity and a reduction of unit updates is not possible. I have also discussed the possibility to further cut down the update precision of the LF method and conducted some experiments to evaluate the influence of different update precision limits in Section 4.2.1 and Section 4.2.2 for two function approximation tasks under several performance aspects. Finally, I found that a precision of $1.0E-5$ is sufficient for the update weights of the LF method. This has resulted then in performance speed ups up to 60% compared with a full update over all units for the sample testbed.

Limitations

Slower Learning Convergence is a major downside of LF. While it is possible to eliminate this disadvantage to a certain degree by applying higher discounts, GF will still be able to perform better in learning environments where data are i.i.d. and that are not prone otherwise to negative interference. This is especially apparent for the Cross function approximation task with a balanced data distribution in Section 4.2.2. Even when applying relatively high discounts, LF is not able to achieve the overall best performance for this test case. For LF, the localization of the forgetting slows down the training process. Another more extreme example is the reinforcement learning task in Section 4.4. Here, the learning trends have been observed for LF with different discounts applied, and it is apparent that higher discounts have achieved a good performance much faster than smaller discounts. Some of the smallest discounts were not able to reach a good performance even after a long training period. This limitation can become a serious problem when available data samples are very limited and fast learning is necessary. This is for example obviously the case for the Mackey-Glass chaotic

time series approximation task in Section 4.3.3, where the limited number of available data samples has forced me to apply very high discounts. But even with the high discounts, it was necessary to additionally apply a very high produce threshold parameter to ensure that units are added early to the model. This is again related to the LF method, because added units are adjusted slowly and it takes more time until it can be decided if new units should be added or not. If I know for an environment that stable learning is possible, then it might be preferable to apply GF instead of LF updates. But otherwise, LF is preferable over GF because of its learning robustness.

5.2 Dynamic Model Selection

In this section, I will discuss benefits and limitations of the dynamic model selection approach applied to *LF(Prop.)*. The discussion is separately conducted for each of the unit manipulation mechanisms and the self-constructing model adaptation.

5.2.1 Produce Mechanism

Benefits

The produce mechanism plays a very important role within the unit manipulation mechanisms as it is responsible for adding new units and learning performance can be highly dependent on it. Although, split is another mechanism that adds new units to the network, it is not able to add units fast enough when new input regions are observed for the first time. On the other hand, the produce mechanism can add new units in the moment that the model is not able to represent the currently observed data sample well enough. This property is especially necessary in a self-constructing model adaptation approach. For the produce mechanism, I have then proposed to take the sum of the input-output-probability over all units as the factor to decide when new units are necessary, ignoring the prior probability $P(i|\theta) = \frac{1}{M}$. By leaving out the prior probability, one avoids that units are added more easily with increasing model complexity. Taking the sum of all probabilities helps to consider not only the unit with the highest probability but also the other surrounding ones. If units are far from the currently observed data sample, then they do not influence the probability sum very much. Therefore, a further localization is not necessary for the produce decision.

Limitations

The produce mechanisms is important but the current decision approach has a big downside: it is dependent on the Gaussian probability density

functions of the input and output space, which have a wide range of possible numerical values. This makes the search of an appropriate produce threshold difficult, since the best value can be found in a large range of values depending on the characteristics of the learning problem. This is for example the case for the MG time series approximation task, where I had to choose a very large produce threshold value to achieve a good learning performance. On the other hand, it might be necessary to choose a very small value for the threshold, which was for example the case for the Lorenz chaotic time series approximation and the reinforcement learning task. The optimal produce threshold value is influenced by different factors, including not only the numerical range of the input and output space but also the amount of noise added, since the output variance is an estimation of the added noise. The output probability density function tends to have smaller values for larger noise levels so that even for the same learning problem different threshold parameters can be preferable depending on the noise. These limitations make the produce mechanism quite difficult to handle and looking into an alternative solution for the produce decision that is easier applicable to learning tasks with limited domain knowledge remains major future work.

5.2.2 Delete Mechanism

Benefits

The new delete mechanism has been shown to be the most important improvement for the proposed method. In Section 3.3.1, I have introduced several possibilities to normalize an accumulated sum of update weights $\langle\langle 1 \rangle\rangle_i(t)$. Then, all three of them are applied and deleting effects are discussed broadly in the experimental results in Chapter 4. The experiments have made apparent that the alternative delete mechanisms with global characteristics possess weaknesses in regard to negative interference.

The first delete mechanism has been applied by *GFnorm*. While here $\langle\langle 1 \rangle\rangle_i(t)$ is normalized naturally by the update method's normalization coefficient, its delete mechanism performs very weakly under certain conditions. *GFnorm*'s delete performance is especially weak when the discount is $\lambda(t) = 1$ and no forgetting occurs. This contradicts the idea of no forgetting since performance should be slower in convergence but stable when no forgetting occurs. Yet, here the *GFnorm* deletion approach provokes the method to behave more and more catastrophically with increasing learning time. I have tested a second type of global normalizer for the *GFdisc* method that is able to perform stabler compared with the *GFnorm* delete mechanism. But both normalizations are based on global information and

have difficulties to perform robustly in negative interference prone environments.

Then, I have proposed a third delete mechanism that employs a normalizer based on the local information of each unit. This delete mechanism is able to perform very robust, and the conducted experiments have shown that the local delete mechanism outperforms the other two. For the Cross function approximation task, additional test cases without forgetting have been considered to compare the different manipulation mechanisms when learning performance is not affected by the parameter updates. Again, local deletions have shown stabler performance. The robustness also translates in the range of applied threshold parameters, where it has been difficult to find stable threshold parameters for the GF methods. Especially for *GFnorm*, it was often necessary to choose very small thresholds that still resulted in many deleted units. The new delete mechanism is therefore not only robuster but also easier to apply since the deletion of units is less influenced by the data distribution and discount schedule than for the two GF methods.

Since the local delete mechanism is more stable, it would be interesting to discuss its applicability to other methods, especially the GF methods. While it is possible to apply it to the GF methods, it is not as naturally done as for LF. For GF, units forget old information at every time step, but the normalizer only counts relevant updates regardless of the amount of received discounts. A localized normalizer applied to GF would likely delay the deletion of units but since the dividend still grows smaller over time due to global forgetting, the deletion behavior is still not as robust as the one observed for the *LF(Prop.)* method. I have not tested this delete approach for the GF methods, but it might be a possible improvement of the GF methods for the application to learning environments where GF is preferable. Overall, I believe that the localized delete mechanism is further improving the robustness of the LF method and an important enhancement, especially in environments where domain knowledge is limited and negative interference a problem.

Limitations

The localized delete mechanism performed very well in the experiments, and I believe that it is the better method for the learning problems considered in the scope of this thesis. Yet, the mechanism is proposed for learning problems where one wants to avoid negative interference and its application is therefore limited to these kind of testbeds. In learning problems where regions without frequent observations are not from interest, for example in case of unwanted outliers, a localized delete mechanism is not the best choice.

5.2.3 Split Mechanism

Benefits

Robustness: The proposed split mechanism shows a robust behavior over the many different experiments that have been conducted. Although, I have evaluated different threshold parameter settings for each experiment, I found no necessity to change the applied threshold parameter for the split mechanism of *LF(Prop.)* since it has performed robustly for all test cases. It should also be noted that these test cases include a broad range of learning situations as well as different grades of noise and data distributions. On the other hand, a similar robustness could not be observed for the static split threshold that is used by the GF methods. Especially for the chaotic time series approximations, it was very difficult to find an appropriate static threshold parameter so that I have applied a dynamic split threshold based on global information about the output variances instead. The selection of the static threshold is highly dependent on the added noise, which makes it difficult to find a parameter that is neither too low nor too high to be effective. Furthermore, I have compared the local split decision with the static threshold for a function approximation task with an imbalanced data distribution, where the local split mechanism has been able to perform favorably over the static one (Section 4.2.2). Overall, I conclude that splitting based on the proposed local dynamic threshold is preferable in learning environments where domain knowledge is limited, since it acts overall robuster and also favorable for imbalanced environments.

Limitations

Computational Complexity has increased for the newly proposed dynamic split threshold compared with a static threshold approach, because nearest neighbor search is applied. For LF, only units with high update weights are changed at each training time step, so it is possible to reduce the computational cost by considering only the splitting of these units. The number of updated units stays relatively constant after an initial learning phase and therefore does not affect the computational complexity too much. On the other hand, each unit's nearest neighbor candidates are searched over the whole network model, so computational complexity increases here with increasing model complexity. For the applied learning problems, this has not affected the computation speed negatively, since model complexity and amount of data samples have been relatively small.

Yet, the increase in computational complexity might become a problem in learning environments that are in need of high model complexities. Then, it would be necessary to consider possibilities to reduce the computational complexity by speeding up the nearest neighbor search. The computational

burden of nearest neighbor search is a well studied problem, and many different approaches have been proposed to reduce the computation time. It is possible to apply data structures to the search, e.g. kd-trees (Bentley 1975). Data structures are then able to save information about each unit's position in the input space so that locally near units are faster to find. When applying data structures to self-constructing model selection approaches, additional properties are necessary because the model complexity is dynamically changing over time. Therefore, it should be relatively easy to add or prune unit information from the data structure, which also means that the data structure needs a certain degree of robustness against changes in its structure. If changes affect the data structure negatively, then the search performance of the data structure decreases and the computational burden increase. In case of kd-trees, a bucket kd-tree would be for example preferable over a simple kd-tree with only one point at the leaf since buckets ensure a better balance in regard to changes of the tree. While the proposed split mechanism is burdened by an increased computational complexity, there are possibilities to reduce the burden by applying for example kd-trees or other data structures.

Performance is not always better for the proposed local split mechanism compared with a static split threshold as some testbeds work better with a global decision. In the experimental results, the static threshold could perform better than the proposed method when data and noises are i.i.d. Actually, it would be possible to improve performance of the local split mechanism by globalizing the dynamic threshold calculation in cases where data are i.i.d., where globalizing means to include more units as neighbors for the dynamic threshold decision. I have however decided to concentrate more on the robustness of the local split mechanism and show its performance throughout all experiments with a fixed parameter for the relative number of nearest neighbors in regard to the model complexity as already shortly mentioned in Section 3.3.1.

5.2.4 Merge Mechanism

Benefits

Model Complexity can be reduced when applying the newly proposed merge mechanism that considers the elimination of redundancies in the network model. For a more detailed comparison of the different model selection approaches for the Cross function approximation task in Section 4.2.2, it was shown that merge is able to reduce model complexities. It probably also functions positively as an opposite to the split mechanism, where one unit is split into two because too much estimation error was accumulated.

Since the split units are positioned next to each other, they might become similar again in the course of learning so that an additional merge is helpful to deal with these unnecessary redundancies.

Performance is improving mostly when merge is applied so that merge does not only help to reduce redundancies but also can affect the learning performance positively.

Limitations

Robustness: While merge is able to improve learning performance in most cases, it has some difficulties in learning environments where data samples are rarely sampled in large regions of the input space. This is for example the case for the Cross function approximation task with an imbalanced data distribution (Section 4.2.2), where as much as 95% of the data have been sampled in less than 2% of the input space. Although, experimental results are not presented in detail in the scope of this thesis, I have previously tested other percentage of imbalance and found that the merge mechanism is able to improve performance for a testbed with up to 75% of imbalance in the data distribution (Backhus *et al.* 2016b). Overall, it might however be necessary to consider high threshold values near to one for merge when the domain knowledge is limited and there is some possibility of merge negatively affecting the learning performance.

Computational Complexity is a problem for the merge mechanism when model complexity increases. The merge mechanism has to find the unit pair with the highest similarity, and in its simplest form all similarities between units have to be calculated each time the merge mechanism is applied. I have already discussed this problem shortly when proposing the mechanism in Section 3.3.1 and one big aim of my new proposal is to make the merge mechanism better applicable in turns of a few hundred time steps instead of every time step. Actually, one can assume that merge candidates do not occur very often and therefore there is no need to check merging candidates at every time step. So, the application of merge can be reduced to a lower frequency, which then reduces the overall computational complexity quite a lot. In my experiments, I have applied merge every 100 time steps, but performance would be unlikely affected very much if application is even more scarce. This depends however also on the overall number of training data.

Furthermore, it is possible to reduce the number of similarity calculations by applying nearest neighbor search. Nearest neighbor search is itself computational heavy, but one can enhance it by employing data structures, for example a kd-tree, in a similar manner as it has already been discussed

for the split mechanism in Section 5.2.3. Then, the similarities are calculated only for a few nearest neighbors instead of all units. This affects the computational complexity positively under the condition that the computational heaviness induced by the model complexity is larger than that of the alternative approach. While merge introduces some additional computational burdens, there are possibilities to reduce this burden and merge itself is a kind of model complexity reduction, which also affects the computational complexity positively.

5.2.5 Self-Constructing Model Adaptation

Benefits

Initialization becomes a lot easier for self-constructing model adaptation, where the network is not initialized before training but adds and prunes new units "on-the-go". It is only necessary to decide on some initial values for the first unit's input covariance and output variance, which I have fixed to the same values for all experiments. The influence of the initial values seems to be relatively low as it mainly concerns the first few units. Self-constructing model adaptation is especially advantageous when domain knowledge is limited, since one can avoid the common network initialization with random values in regard to the numerical ranges of the input and output data of the learning problem.

Reproducibility: When network models are initialized with a certain model complexity before training, model parameters are often chosen randomly in regard to the learning problem. This randomness complicates the recreation of the same learning results in a later test case or the evaluation of why an initialization has performed much better than another one, except all initializations are explicitly stored by the user. On the other hand, a self-constructing model adaptation creates no units before learning and adds and prunes units according to the received data samples. It is then possible to reproduce the same learning results when the same training parameters and data set are used. For example, this has been apparent in the second testbed of the RL task (Section 4.4.3), where no random factor was introduced and learning of the NGnet is discrete. This reproducibility makes it easier to evaluate the effectiveness of different learning parameters for the same data sets without having to worry about the effects of different network model initializations.

Limitations

Convergence to a good learning performance is slower for self-constructing model adaptation because the model is built from scratch. While a randomly initialized model possesses already a certain amount of units that can be directly trained with the observed data, a self-constructing one needs to find an appropriate model complexity first that is able to approximate the problem well. Here, the produce mechanism plays an important role and learning performance can depend a lot on adding enough units in an early stage of training, especially when training data is limited. This has been for example the case for the experiments of the MG time series approximation in Section 4.3.3, where it is necessary to choose a very large produce threshold to achieve a good performance. I have tested other parameters as well and found that a produce threshold of $T_{Produce} = 10.0$ can achieve similar model complexities as $T_{Produce} = 500.0$, but because units are added more slowly the learning performance is much worse. I have already shortly mentioned the same problem as a limitation of LF, but indeed I believe that both LF and the self-construction of the model play an important role in the necessary learning time to reach a good performance. This shows that while a self-constructing approach is easy applicable, it has its limitations in regard to learning speed and therefore can be difficult to apply in learning environments where the number of available training samples is limited.

Chapter 6

Conclusion

6.1 Summary

In this thesis, I considered the application of an ANN with a receptive field based architecture to online machine learning tasks. For online learning tasks, domain knowledge is often limited making a successful application of artificial neural networks (ANN) more difficult than for offline learning. Major challenges include the achievement of robust learning when the problem environment is prone to negative interference and the choice of an appropriate model complexity. Both are challenging for ANNs in general, but this is especially the case in absence of prior knowledge about the environment. I then discussed that ANNs with a receptive field based architecture have a higher potential of performing robust against negative interference because of the local properties provided by the network architecture. Yet, robustness can not be provided only by the network architecture but it is also important to choose appropriate learning and model selection algorithms. In more detail, I considered a Normalized Gaussian network (NGnet) as one ANN with a receptive field based architecture, where the network parameters are estimated online by the EM-algorithm. Recently, it had been proposed to further improve the robustness of the NGnet's learning algorithm by applying localized forgetting to the parameter updates. Global forgetting had been introduced earlier to improve the convergence speed of the NGnet, but its global character make it prone to negative interference.

Based on the previous work, I considered to further improve an NGnet with localized forgetting's applicability to online learning tasks by proposing several changes to the learning and model selection algorithm. My contributions include the revision of the localized forgetting update method, the adaptation of dynamic model selection, improvement of the robustness of the model selection method in regard to negative interference and a discussion about how the characteristics of the localized forgetting method can be used to reduce computational complexity. Dynamic model selection was applied in a self-constructing manner to ease the network initialization problem prior to training, where now it is only necessary to decide

on an input and output covariance of the first unit. Several experiments are conducted to evaluate the effectiveness of the proposed method and compare it with earlier training approaches for the NGnet that apply either local or global forgetting. The experiments showed that the revised localized forgetting method became applicable over the full numerical range of its implied discount factor, while this was not possible before. Also, I considered the application of dynamic model selection to an NGnet with local forgetting for the first time and found it to be performing well in all experiments. In addition, I showed that an NGnet with global forgetting behaves catastrophically under certain training situations, not only because of less robustness of its update method but also because the earlier proposed deletion mechanism is not suitable for dynamic model adaptation, especially in a self-constructing matter. The newly proposed localized deletion approach does not suffer from these problems and performed very well in comparison. Other contributions also showed performance improvements. Overall, the experimental results showed that the robustness of the model selection algorithm, which was not considered before, also plays an important role in achieving an improved learning performance and less proneness to negative interference over a large range of online learning tasks.

6.2 Future Work

6.2.1 Improvement of Unit Production

For the dynamic model selection, new model units are created with a produce manipulation mechanism that bases the NGnet's adding decisions on the representation probability of an observed data sample. Although, this approach is able to construct the network model successfully, the decision on an appropriate threshold parameter is difficult without a trial-and-error study before training. In its current form, the threshold parameter is influenced by many different properties of the underlying learning problem and the possible numerical range of the threshold parameter is very large. For future work, it would be interesting to look into other produce decision approaches that are able to act on a smaller range of possible values and are less dependent on the learning domain so that the applicability is improved in learning environments where prior domain knowledge is limited.

6.2.2 Ease of Threshold Parameter Selection

For the dynamic model adaptation, I applied several unit manipulation mechanism to increase or decrease the network model complexity. This greedy adding and pruning of units is a widely used approach for ANNs, but it comes with the disadvantage that several threshold parameters have

to be set in advance. This is however difficult when domain knowledge is limited as it is often the case for online learning tasks. For my proposed approach, I showed that the behavior of the unit manipulation mechanisms is relatively robust over a broad field of applications for a delete, split and merge mechanism. Yet, selecting a best performing threshold parameter remains a problem.For future work, it would be interesting to further ease the selection of appropriate threshold parameters with good performance by investigating possibilities for automation or by providing guidelines that can support the users in their selection.

6.2.3 Extension of Proposed Ideas to Other ANNs

The applied unit manipulation mechanism with improved robustness showed favorable performance, where especially the newly proposed localized deletion mechanism played an important role against negative interference. For future work, it would be interesting to investigate an extension of the proposed ideas to other ANNs with a receptive field based network architectures that apply dynamic model selection. Depending on the applied model selection approach, similar improvements are likely possible. Furthermore, the online EM-algorithm, which has been used in this work, has been recently applied to a Gaussian Mixture Model (GMM) in Agostini and Celaya (2016) for a RL task. This work also considered dynamic model selection but only applied a produce mechanism to add new units. It did not consider any other manipulation mechanism, making it a constructive only model selection approach. Since the GMM can be described with the same model-based perspective as the here applied NGnet, which was shortly described in Section 2.2.3 and Eq. (2.11), it would be of interest to consider an extension of my model selection approach to the GMM in a first step. Further extensions could be considered for other similar ANN models in a second step.

6.2.4 Improvement of Learning Speed

Over all experiments, it was observable that the localized forgetting method performs robuster and therefore is preferable in online learning tasks. Yet, the learning process slows down due to the localization and the network model needs more time to adapt to the training data. Furthermore, the self-constructing model adaptation further slows down the learning process, because it takes more observations to initially add new units and adjust them accurately. In environments where training data is limited, it becomes then more difficult to achieve good learning performance. For future work, it would be interesting to investigate possibilities to improve the learning speed while keeping the robustness of the localized forgetting method.

6.3 Concluding Remarks

The application of ANNs to online learning tasks is difficult because of limited domain knowledge and proneness to negative interference. Then, robustness is important to achieve a good learning performance and it is influenced by all factors of ANN's learning: network architecture, learning algorithm and model selection approach. Although, negative interference is a well-known problem since many years, further investigations of its mitigation are necessary for most existing ANNs. Most times, these investigations concentrate on the applied learning algorithm and the network architecture. In this thesis, I have showed that further improvement is possible by applying a robust model selection approach in addition to a robust network architecture and learning algorithm. Therefore, robustness against negative interference should be considered for all performance influencing factors to achieve successful learning of ANNs in online learning tasks.

Bibliography

[AC16] A. Agostini and E. Celaya, "Online reinforcement learning using a probability density estimation", *Neural computation*, vol. 28, pp. 1–27, 2016.

[AR05] W. C. Abraham and A. Robins, "Memory retention–the synaptic stability versus plasticity dilemma", *Trends in neurosciences*, vol. 28, no. 2, pp. 73–78, 2005.

[AS95] C. G. Atkeson and S. Schaal, "Memory-based neural networks for robot learning", *Neurocomputing*, vol. 9, no. 3, pp. 243–269, 1995.

[BA09] M. Bortman and M. Aladjem, "A growing and pruning method for radial basis function networks", *IEEE transactions on neural networks*, vol. 20, no. 6, pp. 1039–1045, Jun. 2009.

[Bac+16a] J. Backhus, I. Takigawa, H. Imai, M. Kudo, and M. Sugimoto, "Online EM for the normalized gaussian network with weight-time-dependent updates", in *Neural information processing: 23rd international conference, ICONIP 2016, Kyoto, Japan, October 16–21, 2016, proceedings, part IV*, ser. Lecture Notes in Computer Science, A. Hirose, S. Ozawa, K. Doya, K. Ikeda, M. Lee, and D. Liu, Eds., vol. 9950, Switzerland: Springer International Publishing AG, 2016, pp. 538–546.

[Bac+16b] ——, "Reducing redundancy with unit merging for self-constructive normalized gaussian networks", in *Artificial neural networks and machine learning – ICANN 2016: 25th international conference on artificial neural networks, Barcelona, Spain, September 6-9, 2016, proceedings, part I*, ser. Lecture Notes in Computer Science, A. E. Villa, P. Masulli, and A. J. Pons Rivero, Eds., vol. 9886, Switzerland: Springer International Publishing AG, 2016, pp. 444–452.

[Bac+17] ——, "An online self-constructive normalized gaussian network with localized forgetting", *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. E100-A, no. 3, pp. 865–876, Mar. 2017.

[Ben75] J. L. Bentley, "Multidimensional binary search trees used for associative searching", *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[Ben94] M. Benaim, "On functional approximation with normalized gaussian units", *Neural computation*, vol. 6, no. 2, pp. 319–333, 1994.

[Bis06] C. M. Bishop, *Pattern recognition and machine learning*, 1st edn. corr. 8th printing 2009, ser. Information Science and Statistics. New York, USA: Springer Science+Business Media, 2006.

[BL88] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks", *Complex systems*, vol. 2, pp. 321–355, 1988.

[Bot98] L. Bottou, "Online learning and stochastic approximations", in *On-line learning in neural networks*, D. Saad, Ed., Cambridge, UK: Cambridge University Press, 1998, pp. 9–42.

[Bug98] G. Bugmann, "Normalized gaussian radial basis function networks", *Neurocomputing*, vol. 20, no. 1–3, pp. 97–110, 1998.

[CA15] E. Celaya and A. Agostini, "On-line EM with weight-based forgetting", *Neural computation*, vol. 27, no. 5, pp. 1142–1157, May 2015.

[CCG91] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks", *IEEE transactions on neural networks*, vol. 2, no. 2, pp. 302–309, 1991.

[CMU02] M. R. Cowper, B. Mulgrew, and C. P. Unsworth, "Nonlinear prediction of chaotic signals using a normalised radial basis function network", *Signal processing*, vol. 82, no. 5, pp. 775–789, May 2002.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm", *Journal of the royal statistical society. series b (methodological)*, vol. 39, pp. 1–38, 1977.

[Doy00] K. Doya, "Reinforcement learning in continuous time and space", *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.

[DS06] K.-L. Du and M. N. Swamy, *Neural networks in a softcomputing framework*. London, UK: Springer Science & Business Media, 2006.

[Flo+12] J. P. Florido, H. Pomares, I. Rojas, J. M. Urquiza, and M. A. Lopez-Gordo, "A deterministic model selection scheme for incremental rbfnn construction in time series forecasting", *Neural computing and applications*, vol. 21, no. 3, pp. 595–610, 2012.

[GBD92] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma", *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[Goo15] B. F. Goodrich, "Neuron clustering for mitigating catastrophic forgetting in supervised and reinforcement learning", PhD thesis, University of Tennessee, 2015.

[GP83] P. Grassberger and I. Procaccia, "Characterization of strange attractors", *Physical review letters*, vol. 50, no. 5, pp. 346–349, Jan. 1983.

[Hen10] C. Hennig, "Methods for merging gaussian mixture components", *Advances in data analysis and classification*, vol. 4, no. 1, pp. 3–34, 2010.

[HSS04] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "An efficient sequential learning algorithm for growing and pruning rbf (gap-rbf) networks", *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)*, vol. 34, no. 6, pp. 2284–2292, 2004.

[HSS05] ——, "A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation", *IEEE transactions on neural networks*, vol. 16, no. 1, pp. 57–67, Jan. 2005.

[IS01] S. Ishii and M. Sato, "Reconstruction of chaotic dynamics by on-line EM algorithm", *Neural networks*, vol. 14, no. 9, pp. 1239–1256, Nov. 2001.

[Jac+91] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts", *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[Jac97] R. A. Jacobs, "Bias/variance analyses of mixtures-of-experts architectures", *Neural computation*, vol. 9, no. 2, pp. 369–383, 1997.

[Jai+14] L. C. Jain, M. Seera, C. P. Lim, and P. Balasubramaniam, "A review of online learning in supervised neural networks", *Neural computing and applications*, vol. 25, no. 3-4, pp. 491–509, 2014.

[JJ94] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm", *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[KN93] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks", *Neural computation*, vol. 5, no. 6, pp. 954–975, Nov. 1993.

[KY97] H. J. Kushner and G. G. Yin, *Stochastic approximation algorithms and applications*, New York, 1997.

[Lia+06] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks", *IEEE transactions on neural networks*, vol. 17, no. 6, pp. 1411–1423, 2006.

[Lor63] E. N. Lorenz, "Deterministic non-periodic flow", *Journal of the atmospheric sciences*, vol. 20, no. 2, pp. 130–141, Mar. 1963.

[Low15] D. Lowe, "Radial basis function networks - revisited", *Mathematics today*, vol. 51, no. 3, pp. 124–126, Jun. 2015.

[LSS97] Y. Lu, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function neural networks", *Neural computation*, vol. 9, no. 2, pp. 461–478, Feb. 1997.

[LWY97] R. Langari, L. Wang, and J. Yen, "Radial basis function networks, regression weights, and the expectation-maximization algorithm", *IEEE transactions on systems, man, and cybernetics-part a: Systems and humans*, vol. 27, no. 5, pp. 613–623, Sep. 1997.

[MC89] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem", *Psychology of learning and motivation*, vol. 24, pp. 109–165, 1989.

[MD89] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units", *Neural computation*, vol. 1, no. 2, pp. 281–294, Jun. 1989.

[MG77] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems", *Science*, vol. 197, no. 4300, pp. 287–289, Jul. 1977.

[MHS14] F. Meier, P. Hennig, and S. Schaal, "Local gaussian regression", *CoRR*, vol. abs/1402.0645, 2014.

[Mic84] C. A. Micchelli, "Interpolation of scattered data: Distance matrices and conditionally positive definite functions", in *Approximation theory and spline functions*, S. P. Singh, J. W. H. Burry, and B. Watson, Eds., vol. 136, Dordrecht, Netherlands: Springer, 1984, pp. 143–145.

[Mus+92] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, "On the training of radial basis function classifiers", *Neural networks*, vol. 5, no. 4, pp. 595–603, 1992.

[NFS05] J. Nakanishi, J. A. Farrell, and S. Schaal, "Composite adaptive control with locally weighted statistical learning", *Neural networks*, vol. 18, no. 1, pp. 71–90, 2005.

[Pla91] J. Platt, "A resource-allocating network for function interpolation", *Neural computation*, vol. 3, no. 2, pp. 213–225, Jun. 1991.

[Pow87] M. J. D. Powell, "Radial basis functions for multivariable interpolation: A review", in *Algorithms for the approximation of functions and data*, J. C. Mason and M. G. Cox, Eds., Clarendon Press, Oxford, UK, 1987, pp. 143–167.

[PS91] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks", *Neural computation*, vol. 3, no. 2, pp. 246–257, 1991.

[RG99] V. Ramamurti and J. Ghosh, "Structurally adaptive modular networks for nonstationary environments", *IEEE transactions on neural networks*, vol. 10, no. 1, pp. 152–160, Jan. 1999.

[RN10] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach*, 3rd ed. Upper Saddle River, New Jersey: Prentice Hall, 2010.

[Roj+02] I. Rojas, H. Pomares, J. L. Bernier, J. Ortega, B. Pino, F. J. Pelayo, and A. Prieto, "Time series analysis using normalized pg-rbf network with regression weights", *Neurocomputing*, vol. 42, no. 1, pp. 267–285, 2002.

[SA97] S. Schaal and C. G. Atkeson, "Receptive field weighted regression", *ATR human information processing laboratories, tech. rep. tr-h-209*, 1997.

[SA98] ——, "Constructive incremental learning from only local information", *Neural computation*, vol. 10, no. 8, pp. 2047–2084, Nov. 1998.

[Saa98] D. Saad, "Introduction", in *On-line learning in neural networks*, D. Saad, Ed., Cambridge, UK: Cambridge University Press, 1998, pp. 3–8.

[Sat00] M. Sato, "Convergence of on-line EM algorithm", in *Proceedings of the international conference on neural information processing*, vol. 1, 2000, pp. 476–481.

[SB98] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*. Cambridge: MIT Press, 1998.

[SI00] M. Sato and S. Ishii, "On-line EM algorithm for the normalized gaussian network", *Neural computation*, vol. 12, no. 2, pp. 407–432, Feb. 2000.

[SM96] R. Shorten and R. Murray-Smith, "Side effects of normalising radial basis function networks", *International journal of neural systems*, vol. 7, no. 02, pp. 167–179, 1996.

[SS15] F. Stulp and O. Sigaud, "Many regression algorithms, one unified model: A review", *Neural networks*, vol. 69, pp. 60–79, 2015.

[Tak85] F. Takens, "On the numerical determination of the dimension of an attractor", in *Dynamical systems and bifurcations*, ser. Lecture Notes in Mathematics, vol. 1125, Berlin: Springer, 1985, pp. 99–106.

[Ued+00] N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton, "SMEM algorithm for mixture models", *Neural computation*, vol. 12, no. 9, pp. 2109–2128, 2000.

[VDS05] S. Vijayakumar, A. D'souza, and S. Schaal, "Incremental online learning in high dimensions", *Neural computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[vOW12] M. van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes", in *Reinforcement learning*, ser. Adaptation, Learning, and Optimization, M. Wiering and M. van Otterlo, Eds., vol. 12, Berlin Heidelberg: Springer, 2012, pp. 3–42.

[Wan11] N. Wang, "A generalized ellipsoidal basis function based online self-constructing fuzzy neural network", *Neural processing letters*, vol. 34, no. 1, pp. 13–37, 2011.

[Wat89] C. J. C. H. Watkins, "Learning from delayed rewards", PhD thesis, University of Cambridge England, 1989.

[Wil09] B. M. Wilamowski, "Neural network architectures and learning algorithms", *IEEE industrial electronics magazine*, vol. 3, no. 4, pp. 56–63, 2009.

[WMS95] A. S. Weigend, M. Mangeas, and A. N. Srivastava, "Nonlinear gated experts for time series: Discovering regimes and avoiding overfitting", *International journal of neural systems*, vol. 6, no. 04, pp. 373–399, 1995.

[Wu+12] Y. Wu, H. Wang, B. Zhang, and K.-L. Du, "Using radial basis function networks for function approximation and classification", *ISRN applied mathematics*, 2012, Article ID 324194.

[XJH95] L. Xu, M. I. Jordan, and G. E. Hinton, "An alternative model for mixtures of experts", in *Advances in neural information processing systems*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds., vol. 7, Cambridge, Massachusetts: MIT Press, 1995, pp. 633–640.

[Xu98] L. Xu, "Rbf nets, mixture experts, and bayesian ying–yang learning", *Neurocomputing*, vol. 19, no. 1, pp. 223–257, 1998.

[YGW97] D. L. Yu, J. B. Gomm, and D. Williams, "A recursive orthogonal least squares algorithm for training rbf networks", *Neural processing letters*, vol. 5, no. 3, pp. 167–176, 1997.

[YWG12] S. E. Yuksel, J. N. Wilson, and P. D. Gader, "Twenty years of mixture of experts", *IEEE transactions on neural networks and learning systems*, vol. 23, no. 8, pp. 1177–1193, Aug. 2012.