UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Active Learning in the Sensorimotor Loop

Diplomarbeit

Leipzig, Februar 2005

vorgelegt von: Martius, Georg
geb. am: 25.01.1980

Studiengang: Informatik

Abstract

In this thesis we study a novel approach to on-line learning of artificial neural networks, called backward modelling, and apply it to active learning in the sensorimotor loop.

At first the mathematic foundations of this approach are elaborated. We observe effects like spontaneous symmetry breaking, response increasing, and generalisation improvement at a theoretical level. We then justify the theory with experimental results on some synthetic problems, in order to understand the phenomena clearly. Finally we consider a simple robot with an adaptive world model. In the case the controller of the robot is just covering a subspace of the actuator space we realise degenerated world representations in the world model with passive learning and standard learning algorithms. We show that backward modelling and active learning point out degeneracies in the world model and correct them with direct exploration. A special kind of active learning evolves from the use of backward modelling which directly queries patterns on the fly. Additionally, different strategies are investigated in order to control the interplay of controller based and active learning based behaviour.

Contents

Introduction

The aim of creating intelligent programs is one of the major goals in Artificial Intelligence and Computer Science in general. Learning is the hallmark of intelligence, and many would argue that a system that cannot learn is not intelligent. A system that does not learn cannot be efficient in a changing or partially unknown environment, because it rederives each problem solution and repeatedly makes the same mistakes.

> Learning can be described as normally a relatively permanent change that occurs in behaviour as a result of experience. [1]

Machine learning means that one is looking for a program that enables a machine, i.e. a computer, to perform learning. In this study we will propose a novel approach to machine learning and apply it to artificial feed-forward neural networks. It is called backward modelling. The algorithm proposed here are based on the standard backpropagation algorithm. However, we do not aim for yet another backpropagation improvement, instead we look for some fundamental effects such as spontaneous symmetry breaking and responsive models applicable in situated robotics. The field of autonomous situated robotics is an interesting and promising research area. The current state of the art and the results obtained up to now seem to be somewhat basic, however, a machine or robot which should autonomously act in the real world or which should take part in human life, needs to be extremly adaptive. We believe that the study of basic autonomous robot control is essential for the development of successful robots in real world environments. Motivated by nature, where fresh born creatures have very limited knowledge – given by the genome – about the world and how to control their actuators, we consider the cognitive bootstrapping problem. It arises when the robot learns its controller and its world model simultaneously starting at zero knowledge. Finding general solutions for this problem seems to be very important for building complex and behavioural robust robots. We will analyse a very simple robot and identify a basic problem of degenerated world models. The approach of backward modelling provides a new way to active learning, which can correct these irregularities by direct active exploration.

This thesis is structured as follows.
The first chapter gives an introduction to machine learning with a bias on numerical and

connectionist approaches.
The second chapter follows the line of introduction and summarises the field of active learning. In the third chapter artificial neural networks are introduced with an emphasis on feed-forward networks. Standard learning algorithms such as backpropagation and common modifications are discussed.
At the base of the problems of existing approaches and the idea of homeokinesis we find motivations for the study. In chapter four we present mathematical foundations, implementation details, and experimental results for backward modelling.
In the fifth chapter we investigate the use of backward modelling in the realm of active learning.
The sixth chapter contains the application of backward modelling to the sensorimotor loop with practical results.
Finally we summarise our results and give an outlook to extensions and future work.

Chapter 1

Machine Learning

> Learning denotes changes in a system that enable a system to do the same task more efficiently the next time. – Herbert Simon [2]

In contrast to adaptation which is temporary, learning involves persistent changes in the system. Machine learning means that one is looking for a program that enables a machine to perform learning. We will call that program learner. The input for the learner is usually a training set, that consists of examples from a certain domain in a chosen representation. One distinguishes between supervised and unsupervised learning on whether the training set contains the desired output, the answer of a teacher, or not.

According to Finlay and Dix [1], the typical process of learning follows three phases:

1. Training: a training set of examples of correct behaviour is analysed and some representation of the newly learnt knowledge is stored.
2. Validation: the validation or test set is processed, which may contain new and unknown examples. If necessary, additional training is given.
3. Application: the knowledge is used for responding to some new situations.

These phases may overlap, and the validation phase might be omitted in case the algorithm guarantees some sort of correctness, or the system is learning in a real world environment.

Michalski et al. [3; 4] classify machine learning approaches by the learning strategies they use:

1. Direct input of new knowledge
2. Learning by instruction
3. Learning through deduction

4. Learning from analogies
5. Learning from examples
6. Learning through observation

With increasing number in the above list the participation of the learner in the learning process increases, whereas the influence of the teacher decreases, and the expectation of the learner's performance raises.

Another classification is based on the paradigm used for learning, proposed by Michalski [3] as well:

a) Knowledge intensive and application specific approaches

b) Symbolic knowledge description

- Rule-based learning
- Instance/case-based learning
- Transformation-based learning
- Explanation-based learning

c) Numeric and connectionist approaches

- Genetic algorithms
- Stochastic approaches
- Neural networks (connectionist models)

The two classifications are not independent from each other. Generally speaking, systems of class a and b tend to use strategies 1 to 3, and systems of class c most likely use strategies 4 to 6. The first two classes are covered by areas like Knowledge Based Systems and classical Artificial Intelligence (AI) and should not be stressed here, instead we will focus on numerical and connectionist approaches.

A further significant distinction for machine learning algorithms is whether the order of training samples is significant or not. *Order sensitive* scenarios are more difficult to deal with than *order free* scenarios. Order free environments just require a correct input-output mapping, whereas order sensitive environments force the learner to keep track of the previously seen inputs. The latter can be combined with delayed reward, which is called *reinforcement learning*. The idea is to give a reward if a certain sequence of outputs have been performed or a certain goal was reached, rather than, provide a desired output for each given input. Reinforcement learning is a wide research area, especially in robot control, see [5] for a comprehensive overview.

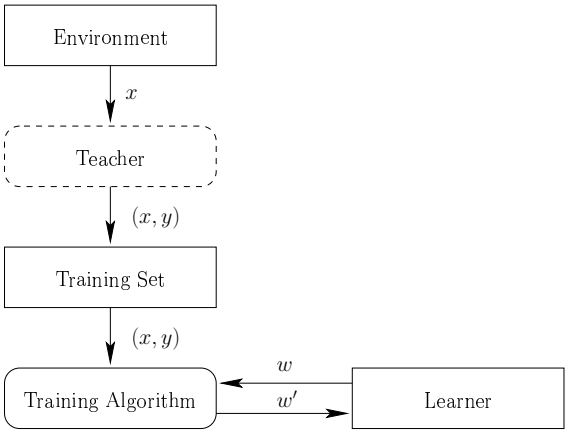

Figure 1.1: Scenario for passive learning

To summarise the learning process we want to formalise the scenario which is sketched in figure 1.1. Please note that the figure just shows the most general case and certain scenarios may well vary. The learner receives input x from an input space $\mathcal{X}$. In supervised scenarios the learner is supposed to map the input to a desired output $y \in \mathcal{Y}$, given by the teacher, whereas in unsupervised scenarios there is no teacher and therefore no desired output. The internal state of the learner will be denoted as $\vec{w}$. The learner can be considered as a function $\mathcal{X} \mapsto \mathcal{Y} : \psi_{\vec{w}}(x) = \hat{y}$. The input space $\mathcal{X}$ is usually assumed to be fixed and known and most cases hold $\mathcal{X} \subseteq \mathbb{R}^p, p \in \mathbb{N}^+$. The samples x are distributed according to some probability distribution $\mathcal{P}$. If the output space is continuous, e.g. $\mathcal{Y} \subseteq \mathbb{R}^q, q \in \mathbb{N}^+$, then one speaks about a regression task, and if the output space is discrete, e.g. $\mathcal{Y} = \{0, 1\}$, the task is classification.

The next sections will briefly introduce some numerical approaches.

1.1 Genetic Algorithms

Genetic algorithms are inspired by evolution. Initially a population of individuals is generated and form the initial generation. All individuals are evaluated with respect to a fitness function and a selection is performed. The process applies genetic operators such as recombination (e.g. cross-over), random mutations, and natural selection to them in order to generate new individuals. After that evaluation and selection is performed again, and the process iterates until an abort criteria is reached[6].

One might ask what that has in common with learning? Let the learning task be encoded in the fitness function. We now consider that the output of the learning system is generated by an individual of the current generation, we realise that the improvement of the individuals appears to be learning.

1.2 Statistical Approaches

Statistical techniques are mostly used to classify the data. Most techniques aim for dimension reduction to make the classification task simpler. They have been proven to be extremely powerful, but they rely on very large data sets. All approaches make some mathematically specified assumptions about the data and have problems dealing with exceptions. Statistical learning has demonstrated great success on many tasks, including speech recognition.

Examples are: Bayesian Learning, Mixture of Gaussians, Factor Analysis, Hidden Markov Model, Maximum Entropy Principle, Expectation Maximisation, Support Vector Machine, and so forth.

Support Vector Machines are a relatively new yet, and are increasing in popularity. SVMs were first proposed by Vapnik and Cortes in [7] for classification tasks. SVMs construct separating hyper-planes between two classes by maximising the margin between these classes in a high dimensional feature space. In other words the inputs are transformed by a nonlinear transformation into the feature space, where the so called support vectors are used for linear classification. The learning is done by appointing the support vectors. SVMs have also been applied to regression tasks [8].

1.3 Artificial Neural Networks

Artificial Neural Networks (ANN), also connectionist models, are inspired by the interconnectivity of the brain. ANNs typically consist of many units, which are simple by itself and which are highly interconnected. The connections between units are weighted and the learning consists of changing the weights to approximate a complex function. The structure of a neural network can also be considered as a directed graph. Usually the network is devided into layers. One distinguishes between input, hidden, and output layers. The input activates the units in the input layer and each of them sends signals to other units, so that they are activated in turn. The activations of units of the output layer represents the result of the networks computation. All remaining units belong to the hidden layers. In chapter 3 we will have a closer look at them.

Chapter 2

Active Learning

2.1 Motivation

Passive machine learning, as described in chapter 1, uses training samples that are produced by a certain distribution. This distribution is fixed over time and implies the following problem, best explained by an example. Let us consider the problem pictured in figure 2.1. The task consists of the classification of points to class C_1 if they lie inside solid ellipse and to class C_0 otherwise. In figure 2.1 the training samples that are available to the learner are marked with their class membership. The dashed ellipse shows the model of the learner, the representation of the classification task, after training with the shown samples. In the left figure one can see large regions where the teacher and the learner disagree (hatched). The generalisation error of the learner can only be reduced, if more training samples from these regions are presented. All other samples will be classified correctly anyway, hence they do not carry new information. The right side of figure 2.1 shows the result after presenting more samples at random. Some samples will be in the hatched region, but with raising performance of the learner, the probability to get an informative sample decreases. This displays a possible inefficiency in the training process. Active learning is supposed to tackle this problem.

The active learning paradigm differs from the passive learning paradigm by the learner's ability to execute actions, that have an impact on the environment. Usually the actions influence the generation of training data. This ability imposes an important challenge for active learning: Which actions should be generated during learning and how can the learner explore its environment efficiently?

The aim of active learning is to reduce the training time by producing concise and highly informative training sets. There are different situations where this can be especially desirable. Learning tasks where training data abound, for example signal processing and speech

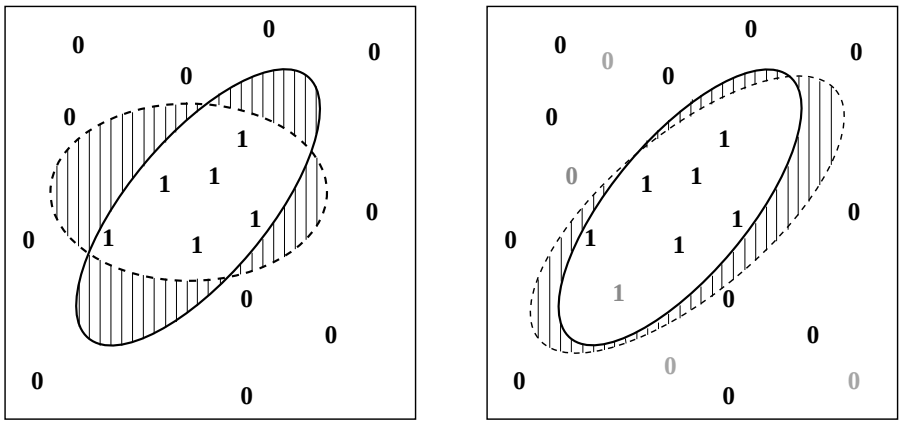

Figure 2.1: Sketch of the generalisation error in a binary classification task. The solid line represents the teacher's classification boundary. The dashed line is the model of the learner. The hatched region stands for the generalisation error (disagree between learner and teacher). *Left*: Representation of teacher and learner after 17 pattern presentations; *Right*: Additional samples (grey) lead to the reduction of the generalisation error. [9].

processing, suffer from the fact that training becomes unnecessarily inefficient. The selection of incisive training data may reduce training time. The contrary situation is given in learning tasks in which training data is sparse and the training data is either expensive or difficult to obtain. For example if training data have to be provided by human experts, which is both slow and expensive. Another scenario is given in dynamic environments, which, for example, a mobile agent will experience. Active learning means here more effective exploration.

2.2 Overview

Let us focus on the differences in the scenarios of active learning in contrast to passive learning. Passive learning consists of a one-way information flow from the environment to the learner as sketched in figure 1.1 on page 5. The learning process takes no account of the state of the learner. Active learning introduces a new flow of information. As displayed in figure 2.2 on the next page the learner gets the ability to perform actions $a \in \mathcal{A}$ with its action generator that depend on its internal state $\vec{w}$. This actions have an impact on the sample generation of the environment.

The central question in active learning is therefore which actions the learner should undertake. The most simple approach [10] is to use random actions. Random action selection is frequently used, because it is simple and ensures that any possible finite sequence of actions will be executed eventually. However, it has been shown, both through theoretical analyses as well as empirical findings, that more sophisticated query and exploration strategies can often drastically reduce the number of training examples required for successful learning. This is

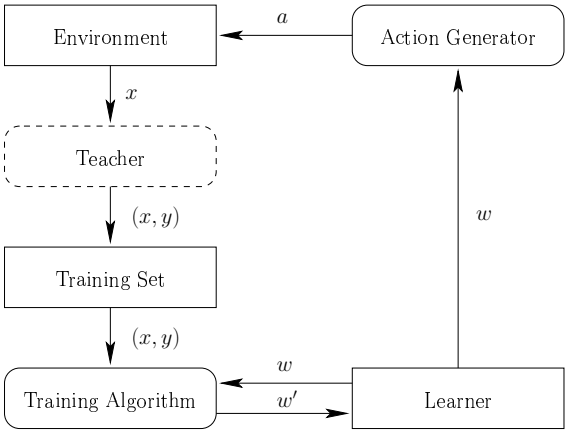

Figure 2.2: Scenario for active learning including the interaction of the learner with the environment through the action generator.

not surprising since different actions mostly produce different training examples which carry different amount of information. Intuitively speaking, in order to learn efficiently one would like to perform actions that produce most informative inputs. The more one expects to learn from the outcome of an action, the better it is. Indeed, this *greedy* principle as been employed in most approaches to action selection.

There are two different criteria to classify active learning algorithms. The first on is whether the actions *select* the training samples from a *random data stream* or from a *fixed pool*, or if the training samples are requested by a *query*. The first approach is related to scenarios where training data exists abound, and the second approach is more likely to be used where training data is sparse or the environment is not static, however this is not necessarily the case.

The second criterion is how the algorithm tries to find the most appropriate action. One distinguishes between *heuristic approaches* and principled approaches based on *optimisation of an objective function*.

Apart from that active learning can be used in conjunction with different classes of passive learning algorithms as mentioned in chapter 1. However, active learning is mostly studied in numeric and connectionist approaches.

Order sensitive scenarios and reinforcement learning are large research areas in active learning, since the interaction with the environment is of primary interest.

The next sections give an overview to different action types as categorised above, and section 2.6 compares heuristic approaches with optimisation approaches.

2.3 Selection

As described in the previous section the actions performed by the learner to influence the environment are characteristic for active learning. In the following sections, we give a summary of different approaches to active data selection via actions.

2.3.1 Concise subset selection

The idea is to select a concise subset from the training set if the training set is very large and possibly redundant. One can assume that not all training samples are equally important to the learning task, therefore the aim of the selection is to remove the training samples that carry the least information. There are two approaches to be distinguished: *growing* and *pruning*.

Growing

Growing-algorithms start with a very small subset and sequentially add training samples to the training set. Plutowski et al. [11; 12] propose such an algorithm. In this algorithm the particular sample is added to the training set if it maximises the expected squared error decrement.

Röbel [13] describes a similar approach that selects samples with the highest predicted error. His algorithm just adds a pattern if over-fitting on the current set occurs. This is calculated with a generalisation factor $\rho = \frac{E_V}{E_T}$, where E_V and E_T are the network errors (e.g. mean square error) of the validation and training sets respectively. Is $\rho \leq 1$ then the sample with the highest error is added. Additionally a random sample is added to the validation set so that both sets contain the same amount of elements.

Zhang proposed in [14] an active selective learning algorithm that adds new samples to the training set if the error is less than a given threshold t. The threshold is given by $t = N_H(N_I + N_O)/\tau$, where $N_{\{I,H,O\}}$ are the number of units in the input, hidden, and output layer. Zhang suggests a value of $\tau \in [100, 200]$.

Another selective algorithm was proposed by Engelbrecht and Adejumo [15], which divides the training set into *large-next-day changes* and *small-next-day changes* subsets. Patterns are more frequently used from the first subset than from the second. After each epoch the subsets are calculated again, where patterns that cause an above average error will be in the large-next-day changes and the rest goes into the small-next-day changes subset.

Engelbrecht and Cloete [16] present an algorithm that is based on sensitivity analysis [17]. Perturbations at the input cause different changes at the output. This algorithm selects

patterns that produce large changes or response.

Engelbrecht [18] compared the above mentioned algorithms. All approaches reduce the training time and mostly the size of the training set. Röbel's algorithm and Engelbrecht's selective algorithm perform similar on perfect data and on noisy data the latter algorithm overtops the others. According to Engelbrechts experiments Zhang's algorithm performs significantly worse.

Pruning

Pruning algorithm cancel samples from the training set. For example in classification tasks Support Vector Machines provide a very natural way for pruning. As mentioned above, SVMs construct separating hyper-planes in a high dimensional feature space. The classification borderline is retrieved by mapping the planes back to the input space in a non-linear fashion. It turns out that only a small number of training samples (support vectors) are sufficient to describe the separating hyper-planes. These samples are highly informative and can be used as a training set. Guyon et al. [19] show that support vectors can by used to select a training set and to clean the data from outliers.

2.3.2 Pedagogical selection

Pedagogical pattern selection aims for finding an optimal training sequence while the number of training samples remains fixed. Online learning, where the learner updates its state after each training sample, is sensitive to the sequence in which the training samples are presented. Munro [20] and Cachin [21] make use of this observation and proposed several heuristics that present pattern with high training error more frequently then those with low training error. The authors found that, depending on the task, the convergence time and the accuracy of the learning process can be improved by such selection strategies.

2.4 Queries

In contrast to selection strategies, which somehow filters the training set, query strategies perform explicit requests. That means the algorithm asks the environment for a training sample at a certain point in the input space $\mathcal{X}$. In case the input space is not continuous the environment will discretise the request and respond with the near most possible point.

There are different approaches to generate queries:

- Internal state exploitation

- Uncertainty model
- Query by Committee

2.4.1 Internal state exploitation

One group of algorithms exploit the internal state of the learner to estimate the expected information utilisation. Obviously it depends strongly on the type of learner how this can be achieved. Usually this is applied in classification tasks where the highest information gain can be expected at the classification boundary, because there the algorithm is most unsure about the classification [22–27].

2.4.2 Uncertainty model

Atlas et al. [28] and Cohn [29] describe two approaches to learning by queries. Both use a neural network to model the uncertainty of the learner. During learning queries are preferred that have the least predictable outcome. Atlas et al. use the difference of two models, that are constructed from the same observations, whereas Cohn analyses the parameters of the estimator. Both algorithms are proven to be superior to random sampling in empirical comparisons. Paass and Kindermann [30] propose to use an external cost function, where queries are preferred that minimise the decision cost, which allows to focus learning on performance-relevant areas. However, their approach is computational expensive, because the use of Monte-Carlo integration.

2.4.3 Query by Committee

Another approach is called *Query by Committee*, proposed by Seung et al. [31; 32], which can be used for classification tasks. A committee of $2k$ learners are trained with the same samples. An input is requested that is classified by the half of the committee to class 1 and by the other half to class 2. By maximising disagreement among the committee, the information gain of the query can be made high. Query by Committee is based on Shannon information theory and the advantage is, that it is independent from the learner's architecture.

2.5 Exploration

Active exploration is used in a completely different setting of active learning, namly where the environment is not static, moreover it is order sensitive. Exploring unknown parts of the environment requires sequences of actions to be executed. Reinforcement learning is usually

used in such situations. The combination of active learning and reinforcement learning is particularly interesting in the sense that both approaches perform actions which have an impact on the environment, either selected by expected reward or by expected information gain. The next sections will shortly discuss some approaches of active reinforcement learning.

2.5.1 Direct exploration

Direct exploration techniques [33] employ models of the expected knowledge gain to direct explorative actions to unknown parts of the environment. The actions are not chosen at random if there is no promising reward, like it is done in normal reinforcement learning. Now actions are selected if they are believed to be informative at the base of the active learning algorithm [34].

2.5.2 Focusing Exploration

Most exploration techniques select actions that maximise the knowledge gain. In order to ensure efficiency of this methodology, two assumptions have been made:

- The heuristic for estimating the gain of knowledge must yield approximately correct action preferences.
- Gaining knowledge must be helpful for the learning task.

In practice both assumptions are not necessarily hold. For example it is often the case that only parts of the environment have to be known in order to perform optimally.

A common strategy to focus exploration is to explore and to exploit simultaneously, by taking both knowledge gain and task specific utility of actions into account. Thrun [35] demonstrated that the combination of exploration and exploitation can lead to faster learning then either component in isolation. However, the ratio between both is a dilemma. Using a dynamically ratio that fades exploration in time is one proposed solution.

2.5.3 Competitive Active Learning

Jürgen Schmidhuber presents two competing embedded agents [36; 37]. Both predict the outcome of an action and bet on their possibly surprising. Each agent profits from outwitting respectively surprising the other. The agents work with the same internal representations of action sequences and evolve models how to predict new internal representations from existing internal representations. The algorithm considers situations as interesting where both agents disagree, regardless whether there are sure or not. Additionally, each agent can veto on actions

it does not consider profitable. Therefore, the system is motivated to focus on those parts of the environment where both agents have confidence but different opinions. If both agents agree on an action the reward for the winner will stay away and the system will shift its focus to novel actions. Concluding, his algorithm focuses on the interesting things by losing interest in both predictable and unpredictable things.

2.6 Heuristic and Optimisation Approaches

Now we want to direct our focus from action selection to the underlying mathematical method to appraise the actions, which is of central interest in most active learning algorithms.

2.6.1 Heuristic Approaches

As the name suggests, these approaches are based on heuristics about the information gain. For example in classification tasks the heuristic is to prefer points in the input space, that lie on or near to the learner's current classification borderline, since the classification of these samples is most likely to be error-prone.

Hence, heuristic approaches need direct access to the internal representation of the learner. In the case of borderline extraction this can be accomplished for various types of learners.

Perceptrons Kinzel and Ruján [26] proposed one of the first approaches to active learning in neural networks using perceptrons. Here the classification boundary is just perpendicular to the weight vector w of the perceptron. Kinouchi and Caticha [38] justified this algorithm on a theoretical basis.

Multi-Layer Perceptrons (MLP or Artificial Neural Networks) It is less obvious how to construct the classification boundary for MLPs. However, Huang et al. proposed in [23] an algorithm that uses the dualism of inputs and weights.

Williams [27], Kindermann and Linden [25], and Jensen et al. [24] elaborated the inversion of a feed forward network with different techniques. The boundary can be obtained by asking for the input while setting the output to the classification boundary, which is perfectly known.

Another way of estimating the informativeness of a pattern can be done using sensitivity analysis proposed by Engelbrecht et al. [17]. They use the network response to estimate the information gain. If small perturbations of the input lead to large changes at the output the pattern is considered more valuable then patterns where the changes are

small. A selection constant is used as a threshold to determine when to include a pattern. The size of this constant in crucial for good performance. [39]

Local Models The query algorithm for local models proposed by Hasenjäger and Ritter [22] draws on the geometrical properties of these models to induce a Voronoi tessellation on the input space. Queries are selected among the Voronoi vertices.

Another approach, already mentioned in section 2.4, is Query by Committee [31; 32]. It can be considered as an optimisation problem, but in most cases the assumptions are not hold and therefore it is considered as an heuristic approach. Query by Committee estimates the expected information gain using a committee of learners. It is originally intended for classification tasks, but Krogh and Vedelsby [40] presented an algorithm for regression tasks that uses the same idea.

2.6.2 Optimisation Approaches

One can also consider active learning as an optimisation problem. Here the utility of an action is formulated as an objective function. Actions for that the objective function attains its global maximum get the highest appraisal and are therefore preferred. An objective functions should include

- the expected information gain
- the expected reduction of the generalisation error.

Note that these two measures are not necessarily equivalent. Furthermore the generalisation error cannot be calculated directly and it is not considered by the majority of proposals, except [38; 41].

MacKay [42; 43] uses Bayesian framework of learning and applied his technique to classification as well as to regression tasks. Cohn [44] minimises the expected output variance of a neural network and Belue et al. [45] proposed an algorithm for multi-class classification tasks with MLPs.

2.6.3 Comparison

Principled approaches are suitable for theoretical analysis, since the optimisation criterion is explicit. The properties of such an algorithm can be obtained more easily than it is the case for heuristic ones. Moreover, the implementation is mostly more difficult and the computational cost is high.

A drawback of heuristic algorithms is that the input distribution is not received by the learner, which can be important according to Eisenberg [46; 47]. Another problem, discussed by Baum and Lang [48], is that constructed samples at the classification borderline can be very difficult to classify for the teacher, in particularly if the teacher is a human. Apart from that the classification borderline may be fuzzy. Overly concentrating on the shape of the line itself may lead to unnecessary complex models. Heuristic approaches are model dependent solutions. The big advantages of heuristic approaches are the easy implementation and the comparable low computation cost.

2.7 Related Work

This section enumerates some recent approaches to active learning, which are difficult to classify in one of the previous sections.

Supervised Training and Unsupervised Clustering

Engelbrecht et al. proposed an unsupervised clustering of the training data combined with a pattern selection approach based on sensitivity analysis [49]. The training data is clustered into groups of similar patterns using Euclidean distance. The most informative patterns are selected from the groups using sensitivity analysis learning. The authors found that the clustering approach improved the performance consistently.

Online Choice of Active Learning Algorithms

Baram et al. [50] combine an ensemble of active learners to expedite learning in pool-based environments. They developed a master algorithm that is based on the known competitive algorithm for the multi-armed bandit[1] problem, and a new semi-supervised performance evaluation statistic. More specifically, they proposed a performance measure for active learning algorithms, called *deficiency*. It is essentially a normalised ratio between the accuracy achieved by the active learning algorithm and by the belonging passive learning algorithm.

The online choice algorithm was found to be more robust and more efficient in a wider range of problems than either of the used active learning algorithms by its own.

[1] The multi-armed bandit problem is named by analogy to the one-armed bandit machine (Colloquial English for gambling machine or slot machine). A gambler has to decide which arm to pull (which machine to use) in order to maximise his total reward in a series of trials.

Chapter 3

Artificial Neural Networks

Artificial Neural Networks (ANN) are a mathematical model of biological neural networks. It is practically a system that consists of a large amount of very simple units, that are compareable to neuron cells, which process information in the form of activations that are transmitted from cell to cell via directed connections.

The following sections of this chapter give a very brief introduction to ANNs, the standard algorithms for supervised learning, and the notations used.

3.1 Neuron / Unit

A neuron is the basic building block of ANNs. To avoid confusion with the biological neuron we will use the name *unit* for it. A unit has n Input connections, an activation function, and an activation or output. Every connection links the output of unit i to an input of unit j and has a weight w_{ij}. The activation, or output, $\hat{y}_j$ of the unit j is calculated as follows:

$$\hat{y}_j = g_j \left(\sum_{i=1}^{n} x_i \, w_{ij} \right) \tag{3.1}$$

where g_j is the activation function of unit j, which describes the behaviour of the unit. Usually one distinguishes between semi-linear and sigmoid units. Figure 3.1 shows typical activation functions.

The biological neuron only sends binary signals. Each signal is called spike and the spike-rate can be considered as an continuous output, which is used in the model. The biological neuron has an axon hillock that acts as a threshold for the signal transmission. One can use a *bias* term, that acts as an input offset, to achieve a similar result. This term is simply added to

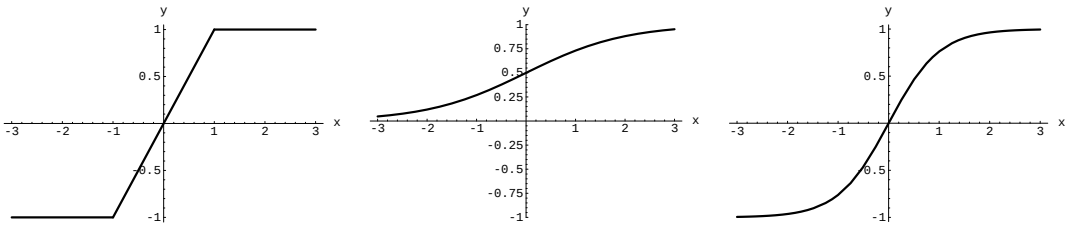

Figure 3.1: Typical activation functions: *Left*: semi-linear with bounds at -1 and 1; *Center*: standard sigmoid $\frac{1}{1+e^{-x}}$; *Right*: tangent hyperbolic

the sum of all inputs before the activation function is applied. An easy way to implement this is the introduction of an additional unit with constant output, the bias-unit. Each unit gets an additional connection to the bias-unit. The weight of this connection is the bias value of the unit.

3.2 Networks

A neural network is a compound of units that are grouped into layers. Layers are a logical constructs that have a more or less specified function. For example, input layers consist of units that get the input from some external source, and output layers produce the result of the networks computation. Hidden layers are layers in between input and output layers. The topology of a network is usually given in terms of layers and their logical connection. However, the real connections are between units.

The input to the network is $\vec{x}$, which is a vector with the length equal to the number of input units. The output is denoted as $\vec{y}$, which is the vector of activations of all output units. In the following, we will omit the $\vec{}$ symbol and write for the network function $\psi : \mathcal{X} \mapsto \mathcal{Y}$:

$$\psi(x) = \hat{y} \qquad \text{for } x \in \mathcal{X} \tag{3.2}$$

3.2.1 Feed-forward Networks

The most simple case of network topology are the feed-forward networks. They consist of input, hidden, and output layers. Connections are just allowed from the input layers towards the output layers, i.e. there are no cycles in the connection graph. The output of the network consists of the activations of all output units. These activations are obtained by propagating the input through the network using equation 3.1 for each unit from the first hidden layer towards the output layer.

3.2.2 Recurrent Networks

Recurrent networks or feedback networks are featured with cycles in the connection graph. One distinguishes between four types of feedback networks:

a) Direct feedback: Units can have a connection from it's output to it's own input. These units tend to stay in the saturation regions since the connection amplifies or inhibits the activation.

b) Indirect feedback: Units from higher layers can have connections to units of lower layers.

c) Lateral feedback: Units within a layer are interconnected.

d) Fully connected networks: Every unit is connected to all other units in the network. However, mostly there is no direct feedback included. (Hopfield Networks)

Widely used forms of recurrent networks are Jordan and Elman networks. Jordan networks have feedback connections from the output layer to an additional input layer, which itself has direct feedback. Elman networks have feedback connections from the hidden layer to an additional input layer without direct feedback.

3.3 Learning Feed-forward Networks

In the following, we want to focus on feed-forward networks together with supervised learning. The learning task in the supervised learning scenarios consists of a good mapping from inputs x to outputs y, which are given by the teacher. The pair (x, y) is called *pattern*.

In general the network's output $\hat{y}$ (3.2) differs from the nominal output y. We define the modelling error as

$$\xi = y - \hat{y} \ . \tag{3.3}$$

The aim of learning is to minimise the modelling error for all presented patterns. We obtain the error function E:

$$E = \sum_{i \in I} (\xi_i)^2 = \sum_{i \in I} (y_i - \hat{y}_i)^2, \tag{3.4}$$

where I is the index set of all patterns. Please note that this is just one possible error function called Squared Sum Error (SSE). Some people use different error functions such as Cross Entropy Error (CE) [51].

Learning in ANNs is performed by adjusting the weights of the connections between the units.

How do the weights w_{ij} have to be changed in order to minimise the error? The most natural way, is to perform a gradient descent on the error function with respect to the weights.

3.3.1 Delta Rule

Let us consider the gradient descent on the most simple network that consists of an input layer and an output layer with linear activation functions.

$$
\begin{aligned}
\Delta w_{ij} &= -\eta \frac{\partial E(W)}{\partial w_{ij}} && \eta \text{ learning rate, } W \text{ weight matrix} \\
\frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial w_{ij}} && \text{chain rule} \\
\frac{\partial E}{\partial \hat{y}_j} &= -2 \cdot (y_j - \hat{y}_j) = 2 \cdot \delta_j && \text{using (3.4)} \\
\frac{\partial \hat{y}_j}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \sum_i \hat{y}_i w_{ij} = \hat{y}_i \\
\Delta w_{ij} &= \eta \ \hat{y}_i \delta_j && \text{factor 2 omitted (moves into } \eta) \qquad (3.5)
\end{aligned}
$$

3.3.2 Backpropagation Rule

If the network has hidden layers one needs to propagate the error backwards from the output layer through the network. The Backpropagation rule [52] is a generalisation of the Delta rule, where the calculation of the δ_j becomes more difficult, but the formula for the weight update stays the same.

$$
\delta_j = \begin{cases} g'(\hat{y}_j)(y_j - \hat{y}_j) & \text{if } j \text{ output neuron,} \\ g'(\hat{y}_j) \sum_k \delta_k w_{jk} & \text{if } j \text{ hidden neuron,} \end{cases} \qquad (3.6)
$$

where g' is the first derivative of activation function g.

The derivation of the Backpropagation rule can be looked up in most books about Artificial Neural Networks, e.g. [53]. The whole Back-Propagation learning algorithm short BACKPROP is sketched in Algorithm 1 on the following page.

3.3.3 Drawbacks of BACKPROP

The BACKPROP algorithm is a straight-forward and general purpose algorithm. However it suffers from general limitations. Since it is a gradient descent algorithm it uses just local

Algorithm 1 BACKPROP for feed-forward networks

```
1  let (x⃗, y⃗) = pattern
2  let 𝒰 = set of all units
3  let (ℐ, 𝒪) = sets of (input units, output units)
4  let (ℒ_H) = set of hidden layer
5
6  activate network (x⃗)                                          feed-forward
7  ∀ j ∈ 𝒪 : δ_j = g'(ŷ_j)(y_j − ŷ_j)                            calculate error at output units
8  foreach L ∈ sort(ℒ_ℋ, Descending) do                          propagate δ values to hidden layers
9     ∀j ∈ units(L) : δ_j = g'(ŷ_j) Σ_k δ_k w_jk
10 od
11 ∀ j ∈ 𝒰\ℐ, ∀w ∈ connections(j) : w := w + η ŷ_i δ_j          update weights
```

information about the error function. The following list enumerates the resulting problems and some solutions.

a) Symmetry: One distinguishes between two situations where symmetry occurs and leads to problems.

   - Symmetry in the network weights: For example if all weights of a certain layer are initialised with the same value, all units of that layer will behave identically.

     Solution: random weight initialisation

   - Symmetry in the environment: If the pattern is symmetric with respect to the current model of the network, which is initially linear for low weights, weight changes cancel each other. This will be discussed in more detail in section 4.4.3.

     Solution: BACKMODEL (with spontaneous symmetry breaking, will be presented in this thesis, see chapter 4 on page 25).

b) Local minima of E (3.4): All gradient descent algorithms have the problem that they are not guaranteed to find a global minimum of E. They can get stuck in a suboptimal solution. In practice BACKPROP finds an acceptable minimum for most problems.

c) Flat Regions: Since the weight updates are proportional to the absolute value of the gradient, the algorithm will make low progress on regions of small gradient.

   Solutions: Momentum term [54], RPROP [55], BACKMODEL (chapter 4 on page 25)

d) Oscillation: If the error surface has a "steep valley", the gradient descent may jump from one slope to the other without reaching the minimum.

   Solution: Momentum term [54], η adaptation , RPROP [55]

e) η sensitivity: The learning rate is a significant parameter for performance of BACKPROP.

It decides the speed of the convergence, and in bad cases the success of the algorithm. The proper selection of η is highly dependent on the problem and the network architecture. If it is chosen too large the algorithm can even diverge.

Solutions: Momentum term, weight update limitation

f) Deep networks: If the network has multiple hidden layers the backpropagated error value becomes very small towards the input layer.

Solutions: PERC [56], BACKMODEL (chapter 4 on page 25)

3.3.4 BACKPROP Modifications and Descendant

There have been made many proposals converning ways to speed up BACKPROP. Mostly domain specific information is used or additional assumptions are made. Some improvements are quite general and widely accepted, whereas others are quite specific.

Momentum Term

BACKPROP with momentum term becomes a conjugate gradient descent and was first described by [54]. The weight update for time step $t+1$ is now dependent on the weight change on time step t, such that

$$\Delta w_{ij}^{t+1} = \eta \cdot (1-\alpha)\hat{y}_i \delta_j + \alpha \Delta w_{ij}^{t} \ , \tag{3.7}$$

where α is the momentum term. It is a simple augmentation which suppresses oscillations and can overcome narrow flat regions. However it is another parameter that has to be chosen.

Flat-Spot Elimination

The weight update of a connection from unit i to unit j is proportional to the error value δ_j for unit j, which is calculated with equation (3.6). If the unit has a sigmoid activation function, which is standard, and is activated such that it is saturated, the first derivative of the activation function is very close to zero. This causes δ_j to be close to zero as well. To overcome this apparent stagnation one can add a small constant to g' [57].

Weight decay

Paul Werbos [58] proposed backpropagation with weight decay. His motivation has been that large weights are biologically implausible and lead to a steep error function which causes

oscillations. He introduced a penalty term to the error function that punishes large weights.

$$E' = E + \frac{d}{2} \cdot \sum_{i,j} (w_{ij})^2 \tag{3.8}$$

$$\Delta w_{ij}^{t+1} = \eta \; \hat{y}_i \delta_j + d w_{ij}^t \tag{3.9}$$

The values of d have to be chosen conveniently, but too large values lead to too small weights in general. However, small weights usually yield a better generalisation performance, but the expressional power of the network is low.

SuperSAB

The SuperSAB algorithm from Jacobs [59] uses an individual learning rate η_{ij} for each weight w_{ij}. The learning rates are adapted to the error surface during training. η_{ij} is increased if the partial derivative $\partial E / \partial w_{ij}$ has the same sign over several steps. If the sign changes n_{ij} is decreased.

Quickprop and Second-Order Backpropagation

The idea of second-order backpropagation is to use the second derivative of the error function to calculate the weight update. Quickprop [57] assumes that the error function is locally quadratic and uses the second derivative to jump very close to the minimum, ideally in one step. However, this assumption is not valid in all cases so Quickprop is mostly very fast but sometimes it will fail through oscillations or because it gets trapped in a very small local minima. Quickprop is an offline learning algorithm, which means that the weight updates are summed up and just applied after a certain amount of patterns are presented. Offline learning is not suitable for real time applications like robot control.

Resilient Propagation (Pprop)

Pprop [55] is a combination of SuperSAB, Quickprop and Manhattan-Trainings, which is like standard backpropagation, but only the sign of the gradient is used for the weight updates. That means that the weight updates are not any longer dependent on the size of the gradient. The learning rate is individual and adaptive for each weight. Furthermore the adaptation will try to invalidate the last update if it realises a sign change in the weight. This stops the algorithm from oscillations. However Rprop, like Quickprop, is an offline learning algorithm.

Backpercolation (PERC)

Backpercolation is a learning algorithm for feed-forward networks developed by Mark Jurik, see [56] in [53]. This algorithm calculates an error function for each unit, instead of one for the whole network. The name was chosen, because it should be able to *percolate* through bumps in the global error function. The main objective of PERC is to overcome the problems of BACKPROP with deep networks. To achieve this, an activation error is calculated for each unit. Jurik uses the duality between the output vector of the previous layer and the weight vector. Therefore, each unit can either change it's incoming connection weights or claim another output from the previous units in order to reduce it's error. The normalised activation error is propagated backwards through the network until the second hidden layer and the weight updates are proportional to the activation error and to the reciprocal of the previous layer output.

According to Jurik and his group, PERC reacts less critically on the choice of the learning parameter λ as BACKPROP on the learning rate η. PERC performs better on deep networks, but the initialisation of the weights is critical. Jurik suggests to initialise each weight of a unit k with either $+\Theta/n_k$ or $-\Theta/n_k$, where n_k is the number of incoming connections of unit k. Θ strongly influences the behaviour of PERC and there is just an heuristical approach to find the appropriate value for it.

In the next chapter we will introduce the central matter of this work, the principle of backward modelling.

Chapter 4

Backward Modelling

4.1 Motivations

Breeding of basic behaviour in autonomous robot control is usually achieved by artificial evolution or learning, where an external objective function is provided, e.g. a fitness function or an explicit learning goal. However, the selection of these functions is very difficult and the evolved behaviours are rather simple like wall following. A common problem is that in order to breed behaviour in a controlled way one has to formulate intermediate goals, which itself have to be chosen carefully to guide the evolution or learning in the desired direction. Certainly natural evolution was not supported this way, instead seemingly goal-oriented behaviours have been produced by completely unspecific principles. Already in 1939 Cannon [60] introduced the principle of homeostasis to understand the goal-oriented reactions of the body in response to external perturbations. His idea is, that a system has an internal state, which is to be kept stationary and the external reactions are just a kind of byproduct of the internal force. This is a very generic principle, but has not found many applications in practice.

Der et al. present in [61; 62] a new approach called *homeokinesis*, which can be considered as the dynamical pendant of homeostasis. Instead of stabilising the internal goal to a stationary state it is now defined by a certain kinetic regime. They use an adaptive self-model of the agent backward in time for the kinetic regime. The misfit between the self-model and the true behaviour is used as a permanently available learning signal.

The self-model is likely to experience symmetric patterns if the, usually symmetric, robot is placed in homogeneous space. For example if the sensors can not detect any obstactles the sensor-space is invariant to translation and rotation. The behaviour of the robot evolves from spontaneous symmetry breaking. However, symmetry can also occurs in traditional learning schemes. Breaking them is usually of advantage. A synthetic highly symmetrical problem is

the XOR or Parity problem, which is studied later.

Another problem of BACKPROP and descendants is the strong dependency on the size of the weight initialisation. In homeokinesis one wants to initialise the weights with very small values in order to avoid any significant preset bias of the network.

Because ANNs are a model of biological neural networks, algorithms should respect biological feasibility and therefore rely on local computation and knowledge only. The standard backpropagation algorithm seems to be biologically quite implausible because of something like equation (3.5) and especially equation s(3.6), which would require the δ value to be propagated backward from the dendrite of the receiving neuron, across the synapse and reverse the axon into the sending neuron and then integrated and multiplied by the strength of the synapse and so forth. However, O'Reilly [63] presented the GENEREC algorithm, which is a biologically more plausible variant of backpropagation. With this in mind we use the standard backpropagation algorithm as a base of our work.

Apart from that, biological networks are usually deep networks and consist of logical subnetworks as well. An algorithm should scale up well on large networks and should support subnetworks and recurrent connections. If the network is confronted with a new environment it should not discard all things learnt before. In other words the current model of the world should be kept as long as possible, however the network should not lose the capability to adapt to completely new things quickly.

In summary the algorithm we are looking for should break symmetry in the inputs, cope with very small weight initialisation, need local computation only, operate fast even on deep networks and respect previously consolidated knowledge.

4.2 Inversion and virtual inputs

In this section we will introduce a method that inverts the network and will generate new inputs for it. We have two incitements for this method. The first one comes from the fact that if one works with, for instance neural network function approximators, the internal units may be in the saturation region. This causes the output function to be very flat and the gradient vanishes so that the learning nearly stalls. Another motivation comes from the paradigm of homeokinesis, where the difference between the model backward in time and the actual behaviour is used as a learning signal. This means one uses the time loop error [64] as the objective function for the learning of a robot controller. Here one needs both temporal directions, forward and backward in time. The backward step is performed by the inversion of the function approximator.

Let us consider an input-output pair (x, y) that is used while training the network ψ. For the

target output y we can write:

$$y = \psi(x) + \xi \ , \tag{4.1}$$

where ξ is the modelling error of the network. Let us consider the inverse operation of reconstructing the input value for the given target output. In other words we want to find $\hat{x}$ that satisfies

$$\psi(\hat{x}) = \psi(x) + \xi = y \tag{4.2}$$

and which is called *virtual input*. Note that in general the reconstruction is not unique, because the mapping performed by ψ is usually over-determined. The obtained value of $\hat{x}$ will depend on the used method, but this seems to be no problem in practice as long as the method stays consistent. The reconstruction can be described as a shift of the input by v such that

$$\hat{x} = x + v \ . \tag{4.3}$$

For the following treatments the smallest shift v that solves the above equation would be most appropriate. However, an exact solution of the problem would be computational expensive and is also not necessary here. The shift may be obtained by descending the error

$$F(v) = \|y - \psi(x + v)\|^2 \tag{4.4}$$

$$\Delta v = \epsilon \frac{\partial \psi(\hat{x})}{\partial x} (y - \psi(x + v))^\top \ . \tag{4.5}$$

This cannot guarantee to find the smallest v, but in most cases it will satisfy this condition if started with $v = 0$. The gradient descent will converge either if $y = \psi(x+v)$ or if $\frac{\partial}{\partial x}\psi(x+v) = 0$ meaning v has reached a point where $\psi(\hat{x})$ has a local extremum, which can be seen in figure 4.1.

If we consider learning with the obtained virtual data points we realise that this would not cause any effect because the error $E(\hat{x}, y) = 0$. Therefore we use partial shifts. Mathematically we can obtain this by introducing a penalty term λ.

$$F_\lambda(v) = \|y - \psi(x + v)\|^2 + \lambda \|v\|^2 \tag{4.6}$$

In practice this has been shown to be quite slow. Another approach is to modify the stopping criteria of the gradient descent. The usual treatment is to stop if v does not change anymore. However, this is not trivial because the slope of ψ can be extremely flat, e.g. if hidden units are in a saturation region. In general, this would require a very small abort-threshold and

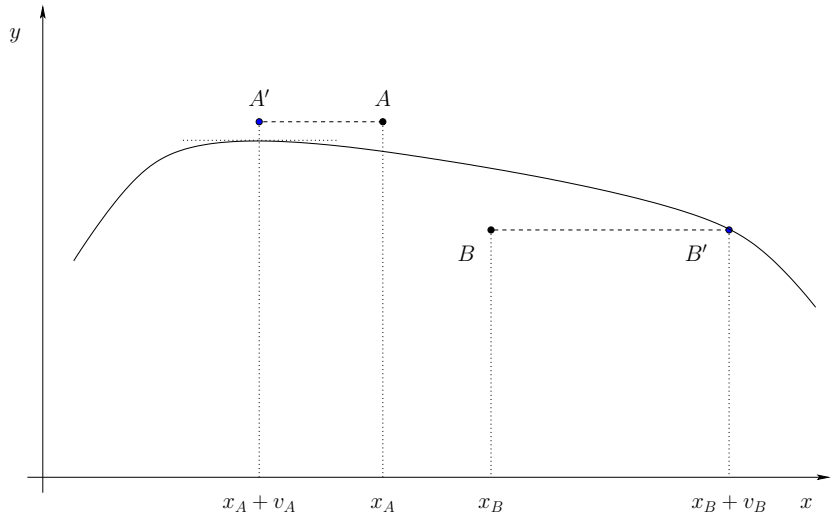

Figure 4.1: Real and virtual inputs. The solid curve represents an arbitrary model ψ(x). We see two data points $A = (x_A, y_A)$ and $B = (x_B, y_B)$. A' is A shifted by v_A to the local maximum of ψ whereas B is shifted to B' which lies on the curve i.e. $\psi(x_B + v_B) = y_B$.

causes many iterations. To reduce the number of iterations we stop the descent if

$$\frac{F(v)}{F(0)} \leq \gamma \tag{4.7}$$

where γ is called the *partial shift ratio*. For a linear model with gradient α the penalty term introduces a shrinking factor of $\frac{\alpha^2}{\alpha^2+\lambda}$, whereas the partial shift ratio will decrease the shift by the factor $1 - \sqrt{\gamma}$ as derived later.

Both approaches are illustrated in figure 4.2, where we see, that the one using the penalty term leads to small shifts in flat regions and to large shifts in steep regions, whereas the early stopping criteria produces large shifts in flat regions and smaller ones in steep regions. The latter one is desired behaviour and will be used in the following.

4.3 Learning with virtual inputs

In this section we want to analyse the effects of learning with virtual inputs, i.e. $(\hat{x}, y)$. To understand the effects, let us consider a simple linear case first.

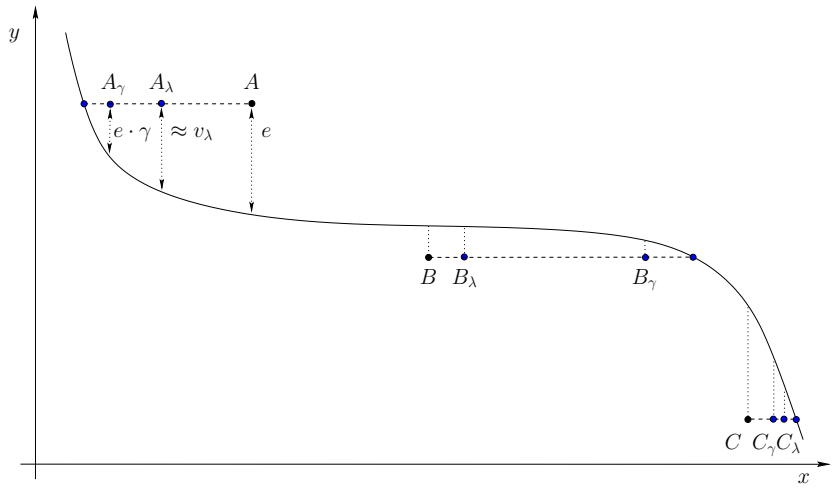

Figure 4.2: Real data points together with fully and partially shifted virtual inputs. The solid curve represents an arbitrary model ψ(x). A_λ is shifted using F_λ with $\lambda = 0.5$ and A_γ is shifted using (4.7) with $\gamma = 0.5$. B and C respectively. We see that the backward method maps points from the flat region to regions where the network is more sensible for inputs and learning.

4.3.1 Linear Case

The model is given by

$$\psi(x) = \alpha x + \beta \ , \tag{4.8}$$

which is just a straight line, and the world is given by

$$y = ax + b + \xi \ . \tag{4.9}$$

In this simple case we can explicitly obtain the shift v from the stopping criteria (4.7)

$$\begin{aligned} \gamma &= \frac{(y - \psi(x+v))^2}{(y - \psi(x))^2} \\ \sigma &= \frac{y - \psi(x+v)}{y - \psi(x)} \qquad\qquad \sigma = \sqrt{\gamma} \text{ because } \gamma > 0 \\ &= \frac{ax + b + \xi - \alpha x - \alpha v - \beta}{ax + b + \xi - \alpha x - \beta} \end{aligned} \tag{4.10}$$

We denote the mismatch of the world and the model parameters with d_a and d_b

$$\alpha = a + d_a, \qquad \beta = b + d_b \tag{4.11}$$

$$\begin{aligned}\sigma &= \frac{-d_a x - d_b + \xi - (a+d)v}{-d_a x - d_b + \xi} \\ &= 1 - \frac{(a+d)v}{-d_a x - d_b + \xi}\end{aligned}$$

$$v = \frac{(1-\sqrt{\gamma})(\xi - d_a x - d_b)}{a+d} \tag{4.12}$$

The partial shift ratio regularises the shift so that for $\gamma = 0$ we have the full shift and for $\gamma = 1$ we get $v = 0$. Let us compare the parameter dynamic of the forward and backward method. The forward model will converge towards the true model, because

$$\Delta\alpha = \eta(y - \psi(x))x = \eta(y - \alpha x - \beta)x \qquad \text{and}$$
$$\Delta\beta = \eta(y - \psi(x)) = \eta(y - \alpha x - \beta)$$

have a stable fix point at $\alpha = a$, $\beta = b$. For the analysis of the fix point $\alpha = a$ we consider α in the environment of a, i.e. $\alpha = a + d_a$, and we assume that β is at its fix point b. For FP $\beta = b$ analogue.

$$\Delta\alpha = \eta((ax+b) - (a+d_a)x - b)x = -d_a\, x^2$$
$$\Delta\beta = -d_b$$

The backward model case contains some surprise since it will not converge to the true model parameter, if noise is included. For the dynamic of α and β we have

$$\begin{aligned}\Delta\alpha &= \eta(y - \psi(\hat{x}))\hat{x} \\ \Delta\beta &= \eta(y - \psi(\hat{x}))\ .\end{aligned} \tag{4.13}$$

If we consider the point $\alpha = a$ and $\beta = b = 0$ then we find $\overline{\xi} = 0$ in $y = \psi(x) + \xi$. We will analyse the averaged parameter update for α:

$$\overline{\Delta\alpha} = \eta\overline{\xi(x+v)} = \underbrace{\eta\overline{\xi}x}_{=0} + \eta\overline{\xi\left(\frac{(1-\sigma)(\xi - d_a x)}{a+d_a}\right)} \qquad \text{see (4.12)}$$

$$= \eta\underbrace{\overline{\xi^2}}_{\geq 0}\left(\frac{(1-\sigma)}{a+d_a}\right) \tag{4.14}$$

$$\Delta\beta = 0$$

and see, that $\alpha = a$ is an unstable fix point. This effect is illustrated in Figure 4.3. For

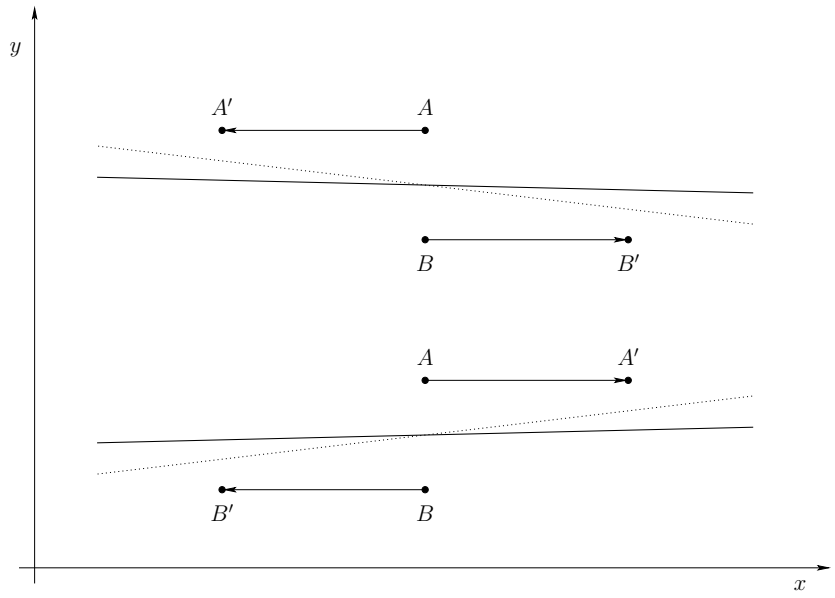

Figure 4.3: The effect of learning with virtual data points. The regression line is rotated (dotted line) even if the presented points would have no effect on the model parameters in the forward model case. The direction of rotation is determined by the initial slope of the model. The diagram is split horizontally to display both cases.

example two data points are presented to the learner that combined effect would not cause any change in the model parameter. Because the virtual data points are shifted to one side or the other, depending on the initial slope, the regression line is rotated.

One might wounder why this is a desired effect that the model does not converges to the right value. To clarify this we want to investigate where the model will converge and why this is of any advantage. For that we define the *response* of the model with

$$R(x) = \psi'(x) \tag{4.15}$$

This notion is based on the fact that the response δy of the model to a variation δx is $\delta y = R(x) \cdot \delta x$. The *response strength* of the model is the absolute value of the response, i.e. $|R(x)|$. In the case currently considered $R(x) = \alpha$. Motivated by equation (4.14) we can say that the backward method increases the response strength of the model in comparison to the forward model. However,the increase of the response strength is counterbalanced by the increasing error between a and α, so that differences in the response strength are not too large. The resulting dynamic can be obtained from equation (4.13). Again, we have $\alpha = a + d_a$ and

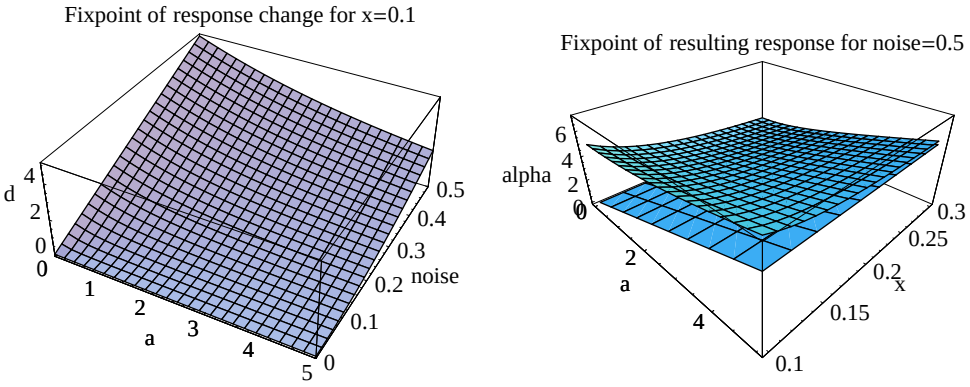

Figure 4.4: Left: Fix points of the mismatch dynamic d in the linear case dependent on the noise and the model response a for x = 0.1. Right: Fix points of the resulting response α dependent on the true response a and the inputs together with the true model (lower plane). Please note that we consider averaged values, which means: noise $=\sqrt{\overline{\xi'^2}}$. Apart from that $\sigma = 0.5$ is used.

$b = \beta = 0$.

$$\begin{aligned}
\Delta\alpha &= \underbrace{(\xi - d_a x - (a + d_a)v)}_{y - \psi(\hat{x})} \cdot (x + v) \\
&= \left((\xi - d_a x) - (a + d_a)\frac{(1-\sigma)(\xi - d_a x)}{a + d_a} \right) \cdot (x + v) \qquad \text{see (4.12)} \\
&= \sigma(\xi - d_a x) \cdot \left(x + \frac{(1-\sigma)(\xi - d_a x)}{a + d_a} \right) \\
&= \sigma \left(\xi x - d_a x^2 + \frac{(1-\sigma)(\xi - d_a x)^2}{a + d_a} \right) \qquad (4.16)
\end{aligned}$$

As before, we consider the averaged parameter update where $\overline{\xi} = 0$.

$$\overline{\Delta\alpha} = \sigma \left(-d_a x^2 + \frac{1-\sigma}{a + d_a} \left(\overline{\xi^2} + d_a^2 x^2 \right) \right) \qquad (4.17)$$

We can see that there is a complicated balance between the force back to the true model (first summand) and the response increasing part (second summand). In figure 4.4 the fixed points of the dynamic are displayed. What we can see is, firstly that the response increase is driven by noice, secondly that small responses are enhanced whereas large responses in the model are kept, and finally that the effect is most visible for small x which are near the symmetry point of the model.

4.3.2 Monotonic Case

A more interesting case is certainly, if the model has the form

$$\psi(x) = g(\alpha x + \beta) \tag{4.18}$$

where g is a monotonic increasing function, where the first two derivatives exist. For instance $g(z) = \frac{1}{1+e^{-z}}$ or $g(z) = \tanh(z)$. In this section we consider the minimisation of v as an objective function instead of the squared output error. We will see below where it can be applied. In the following analysis we are looking for the optimal teacher output. This can be understood as the output y for the input x which lets the model best reduce its error v. In the linear case it would have been $y = \pm\infty$, because there the response of the model is maximal and just small shifts would be required to reduce the output error.

As before we put

$$y = \psi(x) + \xi = \psi(x + v_{full}).$$

and using the Mean Value Theorem we can write

$$\psi(x + v) = \psi(x) + v\psi'(X) \ ,$$

where X lies somewhere between x and $x + v$. Using equation (4.18) we get

$$\psi'(X) = \alpha g'(\alpha X + \beta) = \alpha g'(Z) \qquad\qquad Z = \alpha X + \beta$$

The actual shift can be obtained similarly to (4.12):

$$v = \xi \frac{(1-\sigma)}{\alpha g'(Z)} \tag{4.19}$$

We investigate the minimum of v^2, which can be obtained from

$$\begin{aligned} 0 = \frac{\partial v^2}{\partial \alpha} &= 2v\frac{\partial v}{\partial \alpha} \\ v\frac{\partial v}{\partial \alpha} &= \xi\frac{(1-\sigma)}{\alpha g'(Z)} \cdot \left(-\xi\frac{(1-\sigma)}{\alpha^2 g'(Z)^2}\right) \cdot \left(g'(Z) + \alpha g''(Z)\frac{\partial}{\partial \alpha}Z\right) \ . \end{aligned}$$

We use $g(z) = \tanh(z)$ where $g''(z) = -2 \cdot g'(z)g(z)$ and $\frac{\partial}{\partial \alpha}Z = X$. We get

$$v\frac{\partial v}{\partial \alpha} = -\frac{(\xi(1-\sigma))^2}{(\alpha g'(Z))^3} \cdot g'(Z) \cdot (1 - 2\alpha g(Z)X) \ ,$$

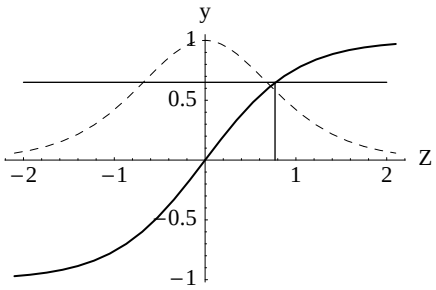

Figure 4.5: Hyperbolic tangent with its first derivative (dashed line) and the optimal value for Z at 0.77 (only the positive solution is drawn).

which is zero for

$$1 = 2\alpha g(Z)X \qquad \text{and}$$
$$1 = 2g(Z)Z \qquad \text{if } \beta = 0.$$

Numerically solved this gives two solutions $Z \approx \pm 0.77$, which lie in the region where the curvature of $g(Z)$ is large (maximum curvature is at ± 0.658). The optimal teacher output is therefore at $y = \tanh(Z) \approx \pm 0.65$, independent of x. That means that training the parameter α with a single data-point x, without considering the true error E, will move the parameter to $\alpha = 0.77/x$, for $\beta = 0$. This can be understood by considering equation (4.19) again. To minimise the shift the denominator $\alpha g'(Z)$ has to be large. Please recall, that $Z = \alpha x + \beta$ and therefore g' is small for large α. Hence, the compromise has been found, which is displayed in figure 4.5. If we consider ψ to be a unit of a neural network with hyperbolic tangent activation function, we see that the weights (here α) are adopted to keep the unit in the region where the slope of the activation function is high and which is essentially also the transition area from nearly linear to saturation behaviour.

In practice we have a set of patterns instead of a single value x. The previous analysis tells us that learning with virtual inputs induces a force that makes the model sensitive on the range of inputs by avoiding the saturation region. Again this force is driven by the model error ξ.

4.3.3 General case

In the general case we obtain the shift from gradient descending the error F from equation (4.4) with the stopping criteria (4.7). The update is following to equation (4.5):

$$\Delta v = \epsilon R(\hat{x})(y - \psi(\hat{x}))^\top \tag{4.20}$$

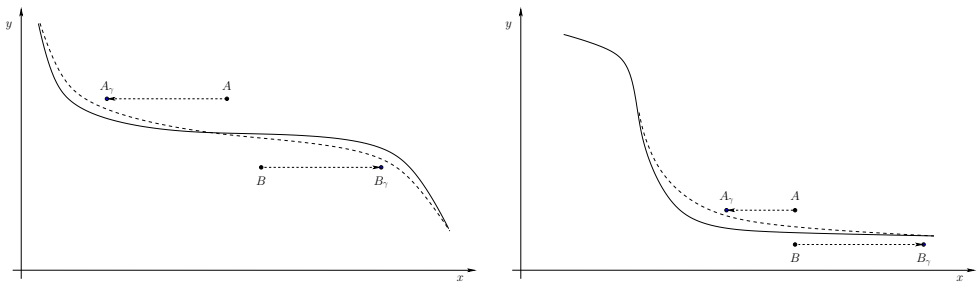

Figure 4.6: Learning with virtual data points lead to response increasing. Data points together with the shifted inputs are shown. *Left*: Both points lead to an increasing of the flat region. The dashed line illustrates the effect. *Right*: A situation which is more likely if the model is a neural network. Point A is shifted towards to responsive region and leads to the increasing of the slope, indicated with the dashed line. Point B is moved towards the saturation region and has a minor effect on the slope.

This dynamic is stationary either if $y = \psi(\hat{x})$ or $R(\hat{x}) = 0$. The first case will not occur because of the stopping criteria. The resulting shift will be

$$v = \frac{\xi(1 - \sqrt{\gamma})}{R(X)} \ , \tag{4.21}$$

where the Mean Value Theorem was used again and X is somewhere between x and $x + v$. We can see that in case of a low model response the shift will be large and vice versa.

Assume a data point (x, y) is given for training, which lies in a flat region of the model. The response strength of the network is weak and the shift is large. It may well happen that the virtual point $\hat{x} = x + v$ is in a region of more response. If we consider the parameter update with the virtual input which is performed by using gradient descent on the error

$$E = \|y - \psi(\hat{x})\|^2 \ , \tag{4.22}$$

we get for an arbitrary parameter p an update of

$$\Delta p = \eta(y - \psi(\hat{x}))\frac{\partial}{\partial p}\psi(\hat{x}) \tag{4.23}$$

where $\hat{x}$ is not necessarily in a saturation region and the model is able to adapt to the given value y at $\hat{x}$. This causes the model to steepen flat regions in the range of presented inputs. In figure 4.6 one can see two cases where the learning with virtual data points increases the response.

We have seen so far, that learning with virtual inputs increases the model response where inputs are presented. This effect is driven by noise and the modelling error. In other words the model gets more sensitive on the region of inputs and particular where it performs badly.

In order to ensure that the model will finally converge towards the true model, the divergence has to be counter balanced by the true error. To achieve that one can perform a traditional learning step with the original input x.

Apart from that, we find this learning scheme leading to two further effects such as generalisation improvement and symmetry breaking. However, before looking into detail we will apply the idea to neural networks and analyse the effects there.

4.4 Application to Neural Networks

In this section we describe the case that the model is given by a feed-forward neural network. Let us see how the error signal can be calculated and the shifts will be obtained. If we consider the most simple feed-forward network without hidden layers and linear activation functions, we can calculate the shift update using equation (4.5):

$$\begin{aligned}\Delta v_i &= \epsilon \sum_j \underbrace{\frac{\partial \hat{y}_j}{\partial x_i}}_{w_{ij}} \cdot \underbrace{(y_j - \hat{y}_j)}_{\delta_j} \\ &= \epsilon \sum_j \delta_j w_{ij} \end{aligned} \tag{4.24}$$

This is essentially the backpropagation rule, see equation 3.6 on page 20. In other words, we just need to use the normal backpropagation rule to calculate the δ values for each unit, but including the input units now. The obtained δ values are used to update the shifts v until the dynamic is stationary or the stopping criteria, equation (4.7), is reached. In the case of non-linear activation functions we have to include the first derivative of g as well.

$$\Delta v_i = \epsilon \delta_i = \epsilon g'(\hat{y}_i) \cdot \sum_j \delta_j w_{ij} \ . \tag{4.25}$$

The update step is the same as for backpropagation, but the network is rather activated with the virtual inputs here.

$$\Delta w_{ij} = \eta \ \hat{y}_i \delta_j(\hat{x}) \tag{4.26}$$

Additionally, we need to perform a weight update for the original inputs as stated in the previous section.

The algorithm is called BackModelGiant because it performs the shift on the whole network. It is sketched at a high level in Algorithm 2 on the next page. The technical issues are covered by section 4.5 and are omitted here to keep the focus on the basics. This method

Algorithm 2 Sketch of the BACKMODELGIANT algorithm for feed-forward networks

let $(\vec{x}, \vec{y})$ = pattern
let $\mathcal{U}$ = set of all units
let $(\mathcal{I}, \mathcal{O}, \mathcal{H})$ = (sets of input, output, hidden units)

activate network $(\vec{x})$ — feed-forward
repeat
  back propagate δ — see BACKPROP (Algorithm 1 lines 7 - 10)
  $\forall\ i \in \mathcal{I} : \Delta v_i := \epsilon \delta_i$
  $\forall\ i \in \mathcal{I} : \quad v_i := v_i + \Delta v_i$ — update shift
  $\forall\ i \in \mathcal{I} : \quad \hat{x}_i := x_i + v_i$ — shift input
  activate network $\left(\vec{\hat{x}}\right)$ — feed-forward
until *stopping criteria eq.* (4.7) = **true** $\vee\ \Delta\vec{v} = 0$
update all weights — see BACKPROP (Algorithm 1 line 11)
BACKPROP $(\vec{x}, \vec{y})$ — perform complete BACKPROP learning step

still suffers from some problems with deep or low responsive networks, which are solved by the layer-wise approach explained in the next section.

4.4.1 Layer-wise shifts

The BACKMODELGIANT might still work quite slow, because the obtained error values δ_i for the input units can be extremely small either if the intermediate weights are small or g' of some hidden units is closed to zero. Both are situations we need to deal with. Small weight initialisation is important for unbiased self organisation effects (see section 4.1) and the first derivative or the activation function is small if the unit is located in its saturation region. Even if there is a small but practical value of δ_i, it would not just require a very long iteration process, it might also produce a very large shift v_i (see 4.21 with small response R). This would not produce the desired effect since other hidden units are likely to reach their saturation region. The idea is to generalise the method to a layer-wise method. Now each intermediate layer will generate a virtual input. The error signal for the corresponding preceding layers is now calculated using the difference between the virtual input and the actual input of the intermediate layer. In other words, the network is considered as a sequence of overlapping 2-layer sub-networks (no hidden layers). The desired output for each sub-network is the virtual input of the subsequent sub-network. The process starts at the output layer and calculates virtual inputs at the direct preceding layer. That means the BACKMODELGIANT algorithm is used at the limited scope of these two layers and updates the weights as well. The obtained virtual inputs (at the last hidden layer) are considered as nominal outputs for these units for the next step, where the next two layers are considered, i.e. the last and the second last hidden layer. This continues until the input layer is reached. Figure 4.7 illustrates

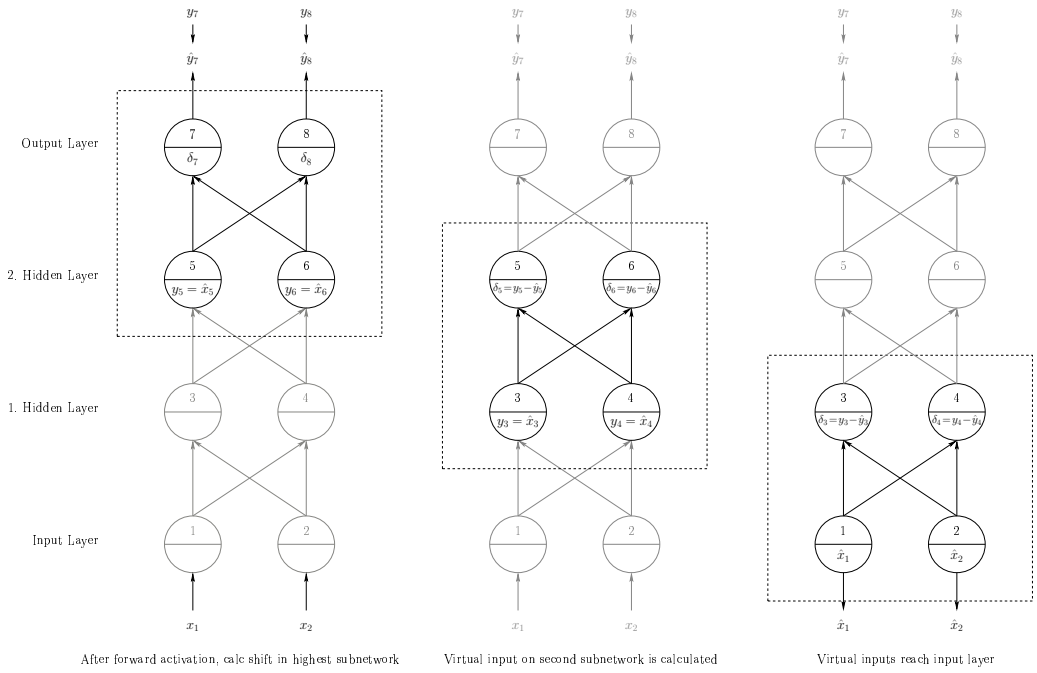

Figure 4.7: Example of layer-wise Backward Modelling (BACKMODEL). Each sub-network (dashed box) is trained using BACKMODELGIANT (see Algorithm 2 on the page before)

the method.

Let us take a look at how the layer-wise approach solves the mentioned problems. The iteration process for calculating v follows equation (4.25), where the δ values are directly obtained from the difference between the nominal and the actual activation:

$$\Delta v_i = \epsilon g'(\hat{y}_i) \cdot \sum_j (y_j - \hat{y}_j) \cdot w_{ij}$$

That means that the calculation of shift update Δv is linear dependent on the size of the weights and on the first derivative of the activation function. In comparison to the multi layer case, where chaining over the layers is used, the shift update is dependent on values in the order of $(\overline{w})^L$ and $(\overline{g}')^L$, where L is the number of non-output layers[1].

Obviously the linearity in the layer-wise treatment will produce a shift in practical amount of steps. Furthermore the resulting shifts are not as large as in the multi layer case, if the weights are small. This means, that the units keep in its operational region and will learn fast. Considering the learning of the intermediate layers we realise, that the objective function is not anymore E (3.4), it is F (4.4) instead. In other words, we use the minimisation of the shift as the objective function. The analysis of the pseudo linear case from section 4.3.2, which is appropriate here, tells us that the model can most effectively minimise the shift if it stays in

[1] $\overline{w}$ and $\overline{g}'$ stand here for the average of the weights w_{ij} and $g'(\hat{y}_j)$ used for the calculation of Δv.

a region of reasonable operability. Applied to the current situation this implies that there is a direct pressure on each hidden layer to build a responsive model and to avoid the saturated region simultaneously.

After we have seen that the ideas of backward modelling can be perfectly applied to neural networks we will have a detailed look at further implications of this approach. The next sections will cover the two general implications namely generalisation improvement and spontaneous symmetry breaking. Apart from that a neural network specific improvement of avoiding dead units is described.

4.4.2 Generalisation Improvement

Generalisation means that the learner is able to interpolate and extrapolate on unknown data points. This requires some kind of continuity in the outputs, because otherwise the generalisation cannot possibly work. Backward modelling makes the learner avoid the saturation region for regular data. A learner trained with other approaches will possibly be already in the saturation region for some regular inputs. At first glance this does not matter as long as the produced answer is correct. However, if we ask the learner for extrapolation, then the chance to be in the saturation region is lower for a learner trained with BackModel. We can therefore expect a better answer.

4.4.3 Spontaneous Symmetry Breaking

Spontaneous symmetry breaking is a very useful property of the backward modelling approach. It was already mentioned, that there are two kinds of symmetry we have to distinguish. The first is the symmetry in the weights of the network, which is equivalent to an unstable fix point in the parameter dynamic. This problem cannot be solved by backward modelling.

The second kind of symmetry can occur in the environment. Symmetry means here, that the parameter updates are symmetric and cancel each other. Obviously this depends strongly on the current representation (model). The interesting case is, if the model, as it currently is, cannot represent the problem and therefore the parameter update will become stationary. For example a linear model in one variable and an axial symmetric environment that yields $y(x) = y(-x)$, e.g. $y = |x|$ or $y = x^2$. Every pair of patterns (x, y) and $(-x, y)$, will cause contrary parameter updates. The sum over all parameter updates will by zero and the model will show no response at all. The only way to brake this symmetry is to make the model more responsive and extend the models capabilities, for example by increasing the polynomial degree of the model.

Let us consider a network with small weights. Even for usual nonlinear activation functions

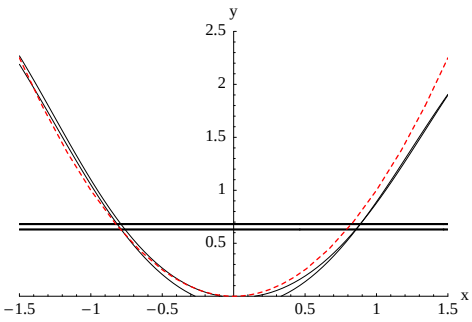

Figure 4.8: Representations for $y = x^2$ for low weight initialisation. We see two randomly picked instances for backpropagation (thick lines) and backward modelling (thin curves). The red dotted line is the true environment.

like hyperbolic tangent, it will behave linearly in the first place. The environment should be described by the parabolic function $y = x^2$ for $x \in [-1.5, 1.5]$. Hence, the environment is nonlinear and axial symmetric. If we train the network with backpropagation we will find it to represent the problem as a constant or flat straight line. This is caused by the fact that the weights are kept small and the algorithm stays in a linear regime.

If we consider backward modelling and recall the response increasing for linear models as discussed in section 4.3.1, we realise that the network would start to increase the gradient of the yet linear model. This causes the weights to raise and the units reach the nonlinear areas of activation. Now the nonlinear dimension of the model comes into play, and the environment is not anymore symmetric with respect to the model. The representation of the environment becomes more precise. The same procedure occurs on higher dimensions, as long as the representational power of the network is not exhausted. In figure 4.8 two randomly picked representations for each, backpropagation and backward modelling, are displayed.

4.4.4 Avoiding Dead Units

One problem with neural networks is that hidden units can become useless, because for example all there incoming or outgoing connections are weak. Because these units have no effect on the outcome of the network one calls them dead units. The usual backpropagation algorithm will just bring them to live again, if either the input or output connections are large enough. In case both are small there is a low activation and the error value is tiny as well. Therefore the weight update vanishes. Backward modelling will, however, create a shift in the unit and activate it virtually. The weight update will be small indeed, but through iteration it will bring the unit to live again.

4.5 Implementation

In this section we focus on the practical implementation of the backward modelling algorithms. It will cover technical details and some practical results obtained from the development and test of the algorithms. We follow the ideas raised through the development process and point to successful and unsuccessful modifications.

The implementation was done in Haskell, which is a pure functional programming language. Please see appendix A for details about the used simulator.

4.5.1 BackModelGiant

In this section we want to specify the over-all algorithm for backward modelling for feed-forward neural networks, short BackModelGiant in more detail. It was already sketched in Algorithm 2 on page 37, however, some practical issues have been omitted.

Real Output Preservation

After a learning algorithm processed one pattern we usually expect that the activation of the output units are actually the outputs of the network to the real inputs. In other words, it should be the same output as with plain feed-forward activation. If we consider the activation of the output units after learning with backward modelling we see that it is the one obtained from activation with the virtual inputs. Therefore, we somehow need to memorise the activation from the first feed-forward step. There are several ways to achieve that. Either we can just store the activation of the output units and restore them at the end, or we can activate the network after training with the original inputs again and perform an additional feed-forward step. The latter version has the advantage of producing proper activations of all units of the network, which might be useful for observation. On the other hand we need to be aware of the fact that it raises the computation cost and the activations are produced using the updated network. In practice this does not matter too much because the weight update is usually small and the extra cost is small in comparison to the overall cost, see section 3.

Additional Backprop Step

An additional learning step with real inputs (standard Backprop) influences the learning behaviour in an advantageous manner, because it will actually take the real error at the output into account. One can think of two forces, one keeping the response in the right range and one reducing the error. Both forces operate simultaneously while learning. The

learning becomes stationary if both forces build an equilibrium. Fortunatelly, the more the model becomes accurate the driving force from backward modelling becomes smaller and the resulting model will converge to the true model.

We could either perform the standard learning step before or after the learning with virtual inputs. As discussed in the previous section there is a need for an additional feed-forward step anyway. Therefore we perform the normal backpropagation learning step at the end. Additionally, this provides us with the true error information δ in the units, which are again useful for observation.

Shift Calculation

The shift calculation introduced the most complicated problems by far. From the mathematical point of view it is a simple gradient descent, with an early stopping criteria. The first naive implementation would be to let the process iterate until either the stopping criteria is reached or the shift update Δv is below a certain threshold. This suffers from a couple of problems:

a) Threshold selection is very difficult. A too large threshold stops already if the gradient is small and a too small threshold leads to many iterations.

b) It requires unknown amount of iterations.

c) The selection of the stepwidth ϵ is crucial for the speed but may cause divergence and oscillation if chosen too large.

In order to find a solution for the problems we need to recall the purpose of the shifts and the required precision. The shifts are used to produce virtual inputs, which are essentially situated more closed to the current understanding of the model than the original input. The partial shift ratio γ determines how much the inputs are transformed into the model and it is supposed to be an uncritical parameter as long as it is in a reasonable range of $0.1 < \gamma < 0.9$. In other words, there is no need to ensure perfectly convergence as long as the resulting effects are not affected.

To solve the problem a) and b), we introduce a maximal number of iterations and stop either if the early stopping criteria is reached or if the maximal number iterations is reached. As a consequence we can statically ensure the maximal computation cost and we do not need to select a threshold value.

The solution for problem c) is a stepwidth adaptation. This should make the selection of an appropriate value for the stepwidth obsolete and in addition it should speed up the gradient descent while avoiding oscillation. For the adaptation we need a measures of the progress and

an adaptation function. The progress-measure conduces as a parameter for the adaptation function. It should provide some information about the current progress and possible divergence and oscillations. One could use some second order approach, but it makes assumptions about the error surface. Instead, we will consider the fraction of the current and the last error $k = \frac{e_i}{e_{i-1}}$. If k is small, the progress is fast, whereas if k is close below 1 the progress is very slow. In case k is larger than 1 or negative we have divergence or oscillation. Let us now consider two possible kinds of adaptation functions:

- exponential: $\epsilon_{new} = \epsilon \cdot e^{\tau(k-\mu)}$, where μ specifies the nominal value of ϵ and τ determines the steepness of the curve.
- parabolic: $\epsilon_{new} = \epsilon \cdot k^{\tau} + \mu$, where μ sets the interval of the factor to $[\mu, 1+\mu]$ and τ influences the steepness and nominal value.

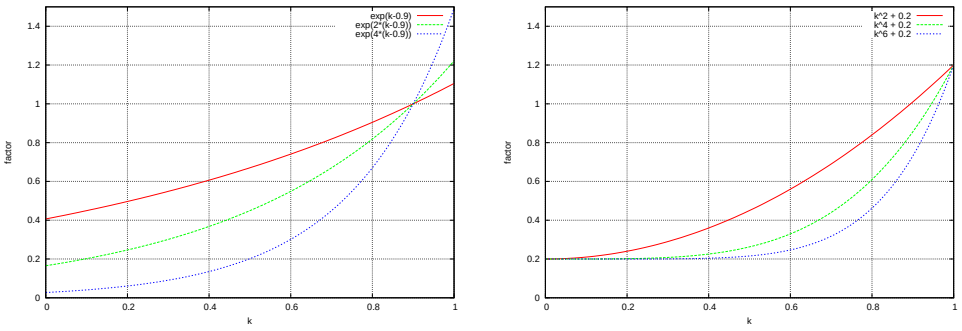

Figure 4.9: Stepwidth adaptation functions. *Left*: exponential; *Right*: parabolic.

One can see in figure 4.9 that the exponential version keeps the nominal value fixed if the steepness is changed, whereas the parabolic function keeps the range of the factor constant. We will use the exponential function with $\mu = 0.9$ and $\tau = 2$. The nominal value of 0.9 means that the error should decrease each step to 90 percent of the previous error. As a consequence we need $i = \frac{\log(\gamma)}{\log(\mu)}$ iterations to reach the early stopping criteria, if the stepwidth is adapted perfectly. However, in reality the error surface might be extremely flat through saturation or low weights and the stepwidth would grow unbound. Therefore, the stepwidth needs to be clipped at a fixed interval. From empirical study we found an interval of $\epsilon \in [0.005, 2.0]$ appropriate. The relatively low maximal bound and the steep adaptation function has been chosen, because of the following case. Let us assume there are many hidden units and each of them is either in the saturation region or has extremely low weights. Consequently the gradient will be small and the stepwidth grows to its maximum. If we leave the saturation region of one or more units now, the gradient raises by magnitudes within a few steps. Therefore adaptation needs be able to decrease the stepwidth very quickly. Comprising, the resulting update rule

for the stepwidth:

$$\epsilon_{new} = \begin{cases} 2.0 & \text{if } \epsilon \cdot e^{2\cdot(k-0.9)} > 2.0 \\ 0.005 & \text{if } \epsilon \cdot e^{2\cdot(k-0.9)} < 0.005 \\ \epsilon \cdot e^{2\cdot(k-0.9)} & \text{otherwise} \end{cases} \tag{4.27}$$

Pseudo Code

A more detailed pseudo code description of the algorithm can be found in Algorithm 3.

Algorithm 3 BACKMODELGIANT algorithm for feed-forward networks

```
1  let (x⃗, y⃗) = pattern
2  let 𝒰 = set of all units
3  let (ℐ, 𝒪, ℋ) = (sets of input, output, hidden units)
4  let max_iter = 100
5
6  activate network (x⃗)                                   feed-forward
7  ∀ i ∈ ℐ : v_i := 0;
8  starterror := E(y⃗)
9  cnt := 0
10 repeat
11       cnt := cnt + 1
12       backpropagate δ                  see BACKPROP (Algorithm 1 lines 7 - 10)
13       foreach i ∈ ℐ do
14          Δv_i := ϵδ_i
15          v_i := v_i + Δv_i                                   update shifts
16          x̂_i := x_i + v_i                                generate virtual input
17       od
18       activate network (x̂⃗)                    feed-forward with virtual inputs
19       e_old := e
20       e := E(y⃗)
21       ϵ := adapt stepwidth (ϵ, k = e / e_old)              see equation (4.27)
22 until e / starterror < γ ∨ cnt > max_iter
23
24 update all weights                  learn with virtual inputs! (Algorithm 1 line 11)
25 activate network (x⃗)                          feed-forward with original inputs
26 backpropagate δ                                  (Algorithm 1 lines 7 - 10)
27 update all weights                               (Algorithm 1 line 11)
```

Cost

In this section the computation cost of the BackModelGiant is compared to the one of Backprop. The complexity class of both algorithms is clearly the same because the shift calculation is a limited loop and introduces a constant factor to the runtime. Therefore we will compare the real computation cost. Let us denote the cost of a feed-forward activation as C_F, the cost of error backpropagation as C_B and the cost of the weight update as C_W. Depending on the implementation of the network the three basic costs may vary, but for simplicity we will assume that they are roughly equivalent and write just c. The cost of one learning step is denoted as C_{BP} and C_{BMG} for Backprop and BackModelGiant respectivly.

$$C_{\mathrm{BP}} = C_F + C_B + C_W \approx 3 \cdot c \tag{4.28}$$

The computation cost for the backward modelling algorithm is not constant, because it depends on the number of iterations i needed for the shift calculation.

$$C_{\mathrm{BMG}} = C_F + i \cdot (C_B + C_F) + C_W + C_{\mathrm{BP}} \approx (2i + 5) \cdot c \tag{4.29}$$

However we can analyse the best, average, and worst case.

Best case: $i = 1: \quad C_{\mathrm{BMG}}^{\mathrm{best}} \approx 2C_{\mathrm{BP}}$

Average case: We may assume that we need twice as much iterations as the optimum of the adaptation process, which means $i = 2 \cdot \frac{\log(\gamma)}{\log(0.9)}$. For $\gamma = 0.5$ we get:
$i = 13: \quad C_{\mathrm{BMG}}^{\mathrm{avg}} \approx 10 \cdot C_{\mathrm{BP}}$

Worst case: $i = 100: \quad C_{\mathrm{BMG}}^{\mathrm{worst}} \approx 70 \cdot C_{\mathrm{BP}}$

However, the cost of providing training examples may be strongly reduced, because of faster learning.

4.5.2 BackModelSim

The BackModelGiant algorithm suffers from the problem, that deep networks and such with low weights lead to many iterations for the calculation of v. The idea of the BackModelSim algorithm is to change the weights and to shift the inputs simultaniously until the error is zero or closed to it. After the learning, the weight changes are taken back partially in order to avoid large weight changes. Therefore, we get a new parameter, that describes the take-back and which takes the role of the learning rate. Biologically one could think of simultaneously changing the synaptic strengths and neuron potential until an equilibrium is reached. After that some kind of pressure on the synapses is released and they relax by taking back most of the changes.

However, the idea has not shown to be very successful. There are two major problems, which are best explained at an example. Let us assume we have a network with very low weights. The loop of weight updates will change them by orders of magnitudes. We also need to keep in mind that the δ values used for the weight updates are the one of the output and hidden units. In contrast, the δ used for the v calculation are the one of the input units, which are almost certainly smaller, especially if the weights are small. Additionally the bias connections catch most of the weight changes, because they are very effective in reducing the error. To avoid that we used a smaller output of the bias unit, e.g. 0.05 instead of 1.0, which decreases the weight update as well.

Back to the example, we realise that the first few presented patterns influence the weights of the network tremendously. Apart from that the resulting shifts are very small and therefore the effects of learning with virtual inputs cannot be exploited. A solution would be to take back a lot of the weight changes, decrease the learning rate, and increase the stepwidth. The situation changes for small initial weights, but not significantly. If the response of the network is high the ratio between the shifts and the weight updates gets higher, which causes more or less a stagnation of learning. In conclusion we can say that the ratio of learning rate and stepwidth decides about the tradeoff between backward modelling and weight update. The take back needs to be large in order to make the network resistant against outliers, but on the other hand it needs to be small to ensure learning if the response of the network is high. This is obviously not a desired behaviour and therefore we consider the BackModelSim algorithm as unsuccessful.

4.5.3 BackModel

Let us now focus on the layer-wise approach as theoretically introduced in section 4.4.1. We will discuss some implementation issues and elaborate three variants of the layer-wise backward modelling algorithm. For that, we will follow the same line as for the BackModelGiant algorithm.

Real Output Preservation

In order to obtain the true outputs one has different possibilities again. Either one memorises the outputs from the first feed-forward step and restores them, or one activates the network with the original inputs at the end. The disadvantage of the first method is that the network has inconsistent activations. Anyway, there is a new issue we need to be concerned about. After the first sub network is trained the next preceding sub network is considered. The error for this sub network is obtained from the difference of the nominal activation (virtual input of subsequent sub network) and the actual activation. That means that each unit needs to

memorise the original activation and the virtual input. The approach used here is that each input unit of a sub network stores its activation before the learning starts and restores it afterwards. At the end the whole network performs a complete feed-forward step, and the activations of all units are consistent and coherent with the input.

Additional Backpropagation Step

The necessity of an additional backpropagation step was not realised initally. The algorithm that does not perform one is called BACKMODELNOBP.

There are basically two different ways to perform the normal learning step, which have a large impact in the learning behaviour. The first one is to use the normal BACKPROP algorithm, which uses the errors back propagated through the whole network. Whether to run it before or after the backward modelling learning step does not make much of a difference. We call this variant BACKMODELFULLBP. The second way is to apply the backpropagation learning step at each layer. The error value used for the weight updates is now the one obtained before starting to shift the inputs at the sub network. In other words, the objection is to minimise v instead of E, which will be shown to be disadvantageous. Nevertheless it is called just BACKMODEL.

How does this reflect in the learning behaviour? The two exclusively layer-wise algorithm have a very similar behaviour. Both optimise the error F (4.4), and the one with backpropagation seems to be more effective in most cases. However, the necessity of the over-all backpropagation step was realised during experiments. The following problem arises through the sole usage of the objective function F. Let us consider a network with at least one hidden layer and small weights trained with some nontrivial environment. At the first glance the learning behaviour looks very promising, because after a few trainings epochs the weights have reached a reasonable size and we find the network to be quite reactive. If we follow the trainings process and assume the network has not yet learnt the task, we observ that the weights to the output units raise and raise whereas the preceeding weights decrease. Clearly this is an undesired effect, but once observed, the explanation is straightforward. The reason has already been mentioned in the theoretically analysis in section 4.3. In order to limit the response increasing one needs a counterpoise, which can be provided by the gradient descent on the true error surface. This is especially important if the model cannot be adapted fast enough. This applies to the currently considered case.

However, in the examples below one can find BACKMODEL and BACKMODELNOBP quite successful. The reason is that in most cases the network learnt the task before the described effect took place. In conclussion the algorithm BACKMODELFULLBP is the one used for further research.

Shift Calculation

The shift calculation is the same as for BACKMODELGIANT because it is essentially the algorithm used for the sub networks. The same stepwidth adaptation process is used, however the calculation will converge more easily since there is just one level of connections between inputs and outputs.

Pseudo Code

A pseudo code description of the algorithm can be found in Algorithm 4. Please note, that the call of BACKMODELGIANT in line 13 can be optimised. The first feed-forward activation step has been performed already. Apart from that the final feed-forward activation is not necessary, and the backpropagation step is not desired in the case of BACKMODELNOBP and BACKMODELFULLBP.

Algorithm 4 BACKMODEL algorithm for feed-forward networks

```
 1 let (x, y) = pattern
 2 let U = set of all units
 3 let I = sets of input units
 4 let (L_I, L_H) = set of input layers, set of hidden layers
 5
 6 activate network (x⃗)                                          feed-forward
 7 ∀ i ∈ I : v_i := 0;
 9 foreach L ∈ sort(L_O ∪ L_H, Descending) do                     consider sub-networks
10   let L^{L-1} = set of preceding layers of L
11   ∀i ∈ units(L^{L-1}) : store(x_i)                             memorise input
13   BACKMODELGIANT on subnet(L, L^{L-1})     L is output and L^{L-1} are input layers
14   ∀i ∈ units(L^{L-1}) : y_i = x̂_i                             set nominal value
15   ∀i ∈ units(L^{L-1}) : restore(x_i)                          restore input
16 od
17 activate network (x⃗)                                  feed-forward with original inputs
18 comment: the following only if BACKMODELFULLBP is used
19 backpropagate δ                                          (Algorithm 1 lines 7 - 10)
20 update all weights                                       (Algorithm 1 line 11)
```

The cost for the algorithm is roughly the same as the one of BACKMODELGIANT and should not be considered here again.

4.6 Experimental Results

This section shows the experimental results obtained from different scenarios. We will consider the XOR and the n-Parity problem mainly to analyse the effects of symmetry breaking and low weight initialisation. After that the results from the Encoder problem will be discussed in order to justify coping with deep networks. Finally the nonlinear regression task, Square is considered to see the generalisation performance and the effect of noise and outliers.

Before taking a look at the actual results we need to clarify the used training process.

4.6.1 Training

There have been done a lot of work on how to perform neural network training simulations and how to generate comparable and valid results [65; 66]. We will follow some of the given guidelines, however, we do not aim for another backpropagation improvement in the general sense. Prechelt [66] claims to use real world datasets in order to measure the real performance of an algorithm. As stated before, we want to analyse certain effects of backward modelling and these can be studied best on synthetic problems. We will consider the application to robot control later.

The following terminus will be used: A *pattern* consists of input and nominal output. A *dataset* is a set of patterns and, a *trainings epoch* is the training of a network with patterns sampled from a dataset according to some *input distribution*.

The three major questions we have to answer are how to present the patterns to the network, when to consider the training as finished, and what are the desired measures. The answers are dependent on the kind of the learning task (classification or regression) and the type of data (perfect data or noisy data with outliers). Let us investigate the noisy case first.

Real World Data

Here fixed datasets are considered only. Obviously, for real robot control we face real-time interaction, which requires a different setting. The measures we are interested in are the generalisation performance and the mapping error. Real world data is usually noisy and contains outliers. In order to achieve a suitable environment we split the dataset into three distinct sets [66]: *Training set*, *validation set* and *test set*. As the name implies, the training set is used to train the network. After one or more training epochs the network is validated using the validation set. The result is used to determine when to stop the training process. The test set is finally used to obtain the measures. Please note, that the network has not seen the test set before.

The termination of the training process is not trivial and is discussed in [67]. The general idea is to prevent the network from over fitting to the training set. If this occurs the error on the validation set will raise, instead of decrease. However, it is not trivial to detect when such a minimum is reached since the future is unknown and one could easily stop at a local minima. We will use the generalisation lost (GL_α) with the modification to use a memory of some previous epochs to jump over small bumps. The generalisation lost at epoch t is specified as the relative increase of the validation set error over the minimum so far.

$$GL(t) = \left(\frac{E_{Val}(t)}{\min_{t'<t} E_{Val}(t')} - 1 \right) , \tag{4.30}$$

where $E_{Val}(t)$ is the validation set error at epoch t. The stopping criteria GL_α says that the training should be stopped after the first epoch has a generalisation lost larger than α. We claim that k previous epochs have a larger generalisation lost than α. Let us call the stopping criteria GL_α^k.

For regression tasks one will usually measure the averaged error over all patterns from the test set. For classification tasks we can calculate the classification error, which is the percentage of wrong classified patterns.

Perfect Data

If we have a perfect synthetic problem such as XOR, where all pattern have the right label, we need to distinguish between classification and regression tasks. Let us start with regression tasks. The validation, like in the noisy environment, is not necessary because over fitting cannot occur. A better stopping criteria might be a threshold for the sum of the errors for each pattern of an epoch. Since the data is perfect an error close to zero must be possible and a fixed threshold seems reasonable.

For a classification task we probably want the network to classify all patterns of the training set correctly. As an example, consider a two class classification problem and a network with one output unit. We can associate the first class with the outputs < -0.5 and the second class with the outputs > 0.5. Everything in between is neither of both classes.

The measure obtained from this kind of simulations is usually the number of iterations needed to accomplish the learning task. In case a test set is used the generalisation performance and classification error can be calculated as well.

4.6.2 XOR and n-Parity

The XOR problem consists of the mapping from inputs to outputs according to the well known "exclusive or" function from boolean algebra. The generalised version of the XOR function is the so called n-Parity function, where n is the dimension or the number of input variables, hence XOR is equal to 2-Parity. Parity is defined as $p(x) = \text{odd}(|\{x_i \mid x_i = \text{True}\}|)$, which is True if and only if the number of True inputs is odd.

Obviously these problems are classification problems with perfect data. Each pattern carries significant information and a split into training and validation set would not make sense here.

The reason why we study the n-Parity problem is that it is highly symmetrical and needs an n dimensional separator. We want to proof the assertion made in section 4.4.3 that backward modelling can deal with symmetric problems.

Network

We use a "recoding" network with one hidden layer for the n-Parity problem. The specification:

- 1 Input Layer: n units with linear activation function and no bias
- 1 Hidden Layer: h units with tanh activation function and bias
- 1 Output Layer: 1 unit with tanh activation function and bias

The network has all feed-forward connection between adjacent layers. Connections have a minimal absolute weight of 10^{-4} to avoid dead connections. The first derivative of the hyperbolic tangent function is lifted by 0.05 to ensure flat-spot elimination as described in [57].

The input is -1 for False and 1 for True and the nominal output is -0.9 and 0.9 respectivly. The tolerated error is 0.4, i.e. an output < -0.5 is considered as False and an output > 0.5 as True.

The number of units needed to solve the n-Parity problem in theory was discussed by many [68–70]. However, to expect from a learning algorithm to solve the task with a minimal network is of questionable value, because in real applications we do not know the minimum architecture and we will probably choose a larger network anyway. For the following experiments we will use $h = 1.5 \cdot n$ if not differently stated.

Parameters

In order to be able to compare the different algorithms we need to consider the whole or at least a large range of the parameter space. There are parameters of the algorithm itself

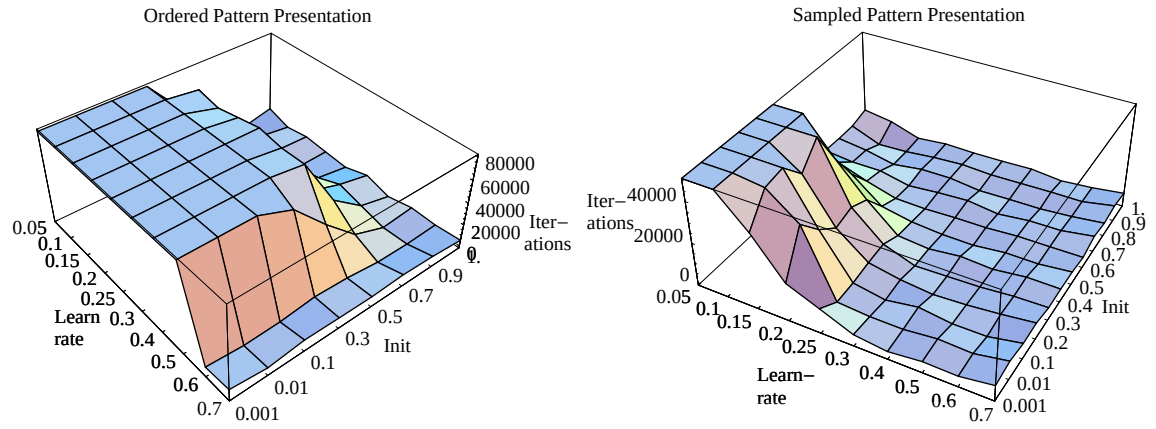

Figure 4.10: BACKPROP: Dependency on the input distribution for 4-Parity. The simulation was aborted after 80 000 (left) and 40 000 (right) iterations. *Left*: ordered pattern presentation; *Right*: patterns sampled. The results for shuffled pattern presentation lie somewhere in between.

and parameters of the training environment, which are mostly not stochastically independent. We ran 25 independent simulations for each parameter setting and collected the number of iterations (pattern presentations) needed for accomplishing the task. That means the training was stopped if all patterns from the dataset have been classified correctly.

BACKPROP

First of all, we want to study the standard backpropagation algorithm on 4-Parity with six hidden units. Let us start with the dependency on the weight initialisation. If we initialise the weights within a small interval around zero the network behaves linear. The weight changes resulting from the patterns are symmetrical and the network will not be able to learn the task. The only way for BACKPROP is that one learning step brings the hidden units into the nonlinear region of activation. This will eventually happen if the the learning rate is very high or one pattern is presented a few times in a row. Therefore, we should see a large dependency on the input distribution. The following distribution have been used:

- Ordered pattern presentation: Each epoch all patterns from the dataset are presented in the same order.
- Shuffled pattern presentation: Each epoch the patterns from the dataset are shuffled (each pattern presented once).
- Sampled pattern presentation: The patterns are randomly sampled from the dataset.

Figure 4.10 shows the expected result, that for ordered pattern presentation the initialisation and the learning rate needs to be very large. For sampled pattern presentation, the

requirements are still valid, but less significant.

Why do we care about the learning rate? The learning rate determines how much the network changes its weight every learning step. If it is chosen too large, then two things can happen. First, the gradient descent becomes unstable and therewith the learning will either not converge or will jump around. Second, the presentation of the mapping function, the model, will be fragile. That means that if a wrong pattern is presented to the network then the model can by heavily effected or even destroyed. Therefore the learning rate needs to be chosen high enough for a learning success in reasonable time, but low enough to make the learning process smooth and stable. A learning rate of $\eta = 0.1$ is usually a good choice for nonlinear networks, but it might vary.

BackModelGiant and BackModel

Let us now take a look at the backward modelling algorithms. In section 4.3 the effects of response increasing and spontaneous symmetry breaking have been discussed in theory. We will see now, that the practical results confirm the theory. The backward modelling algorithms have the partial shift ratio γ as an additional parameter. The partial shift ratio decides about the size of the shifts, hence we can investigate the direct effect of the learning with virtual inputs.

First, we will take a look at BackModelGiant on 4-Parity, because for $\gamma = 1.0$ it is essentially the same as Backprop with the only difference that for each pattern the weight update occurs twice. Figure 4.11 shows the needed number of iterations in dependency on the partial shift ratio and the learning rate with ordered pattern presentation. The weight initialisation was chosen in the interval $[-0.1, 0.1]$ (init=0.1). We can see that the learning with virtual inputs increases the performance significantly. The selection of the partial shift ratio is fairly uncritical as long as in the interval $[0.1, 0.7]$, and we will use a value of $\gamma = 0.5$ except stated differently. In figure 4.12 the behaviour of BackModelGiant is shown in respect to the weight initialisation and the learning rate. As predicted earlier, the overall approach has still problems with low weight initialisation. This is because the backpropagated error is vanishing small.

Now we will look at the BackModel algorithm and its behaviour in dependency on the partial shift ratio. For a partial shift ratio of 1.0 we should not get any successful result, because the hidden layer learns just with the virtual inputs. As we will see, this is not the case because one step of the gradient descent algorithm is performed anyway. Eventually the shift is already large enough to produce weight changes. Figure 4.13 shows the dependence of the partial shift ratio and the learning rate with ordered pattern presentation. Again, the selection of the partial shift ratio γ should be somewhere in the interval $[0.1, 0.9]$ and $\gamma = 0.5$

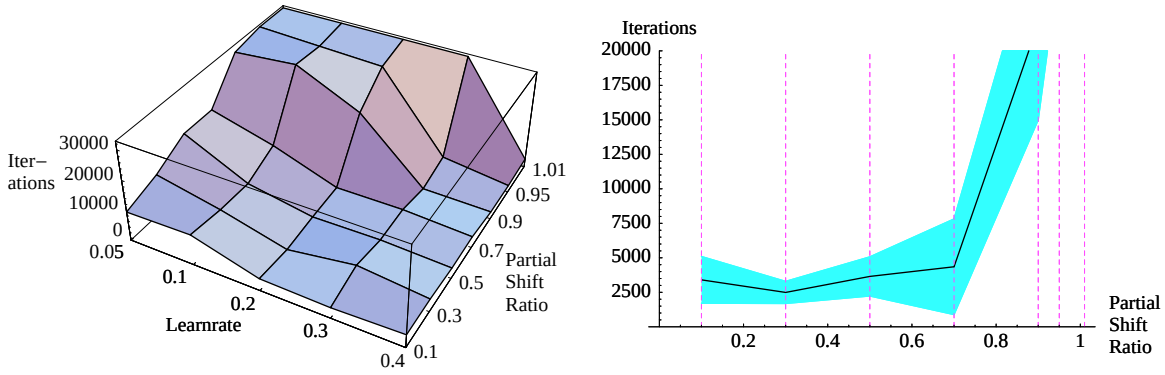

Figure 4.11: BACKMODELGIANT: Influence of the partial shift ratio for 4-Parity. Parameters: maximal iterations: 32 000, ordered pattern presentation, init: 0.1 (random initialisation in the interval $[-0.1, 0.1]$). *Left*: Learning rate - Partial shift ratio - Iterations; *Right*: Slice of the left diagram at learning rate $\eta = 0.2$. The cyan (dotted) vertical lines mark the used parameter value. That means the simulation was performed for $\gamma = 0.1, 0.3, 0.5, 0.7, 0.9, 0.95$, and 1.0. The thickness of the blue (shaded) hose around the black line indicates the standard deviation.

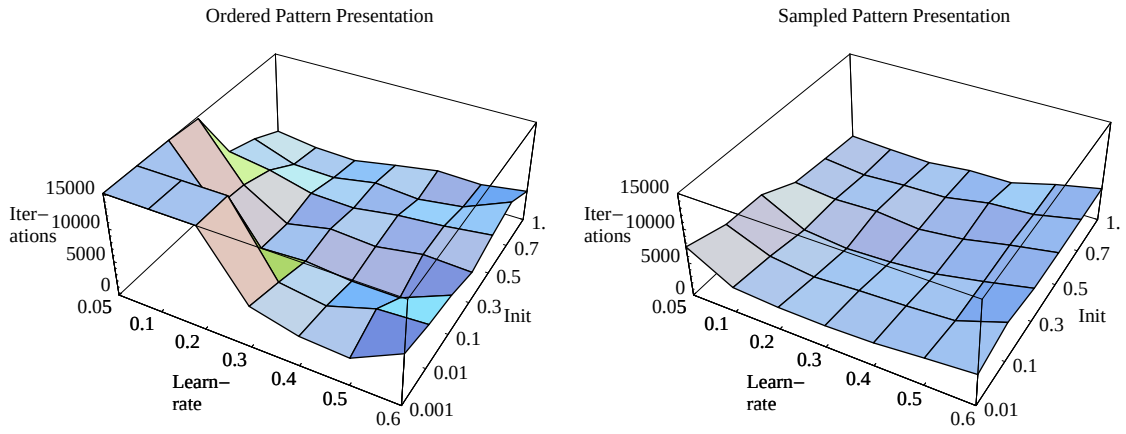

Figure 4.12: BACKMODELGIANT: Dependence of the input distribution for 4-Parity. The simulation was aborted after 32 000 iterations. *Left*: ordered pattern presentation; *Right*: patterns sampled.

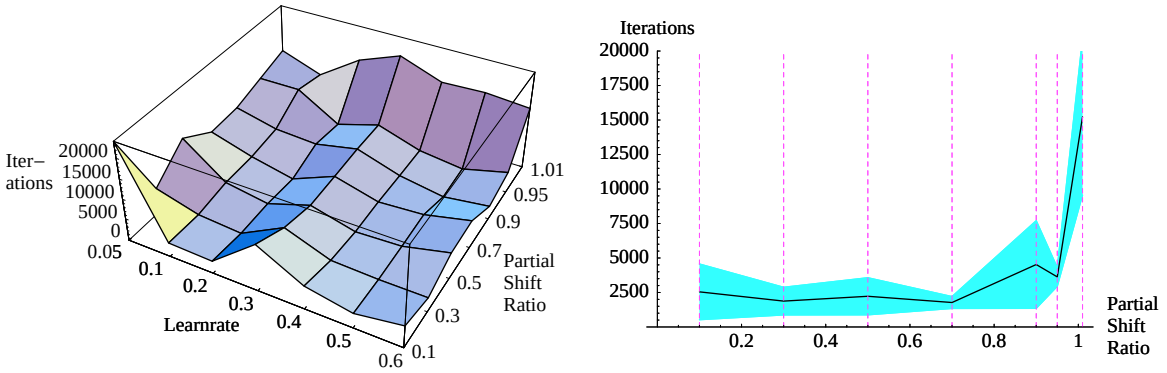

Figure 4.13: BACKMODEL: Influence of the partial shift ratio for 4-Parity. Parameters: maximal iterations: 32 000, ordered pattern presentation, init: 0.1. *Left*: Learning rate - Partial shift ratio - Iterations; *Right*: Slice of the left diagram at learning rate $\eta = 0.2$. The cyan (dotted) vertical lines mark the used parameter value. The thickness of the blue (shaded) hose around the black line indicates the standard deviation.

will be used as a default. The behaviour of the algorithms in respect to the selection of γ was studied on other problems as well, with the same result. Figure 4.14 displays the performance of BACKMODEL in respect to learning rate and weight initialisation.

Comparison

After we have seen that learning with virtual inputs has a positive effect, let us compare BACKMODELGIANT, BACKMODEL and BACKPROP directly. Figure 4.15 shows the results for different learning rates. The weight initialisation has been chosen small ($[-0.01, 0.01]$). For ordered pattern presentation, which holds the most symmetry, the results are remarkable rich in contrast. The standard backpropagation algorithm is only able to solve the problem with extremly high learning rates, whereas BACKMODELGIANT already gets it with the half of that. The layer-wise approach (BACKMODEL) is vast superior here and performs the task even for low learning rates. However, for sampled pattern representation, we see that the BACKMODELGIANT algorithm is very insensitive to the selection of the learning rate.

The influence of the weight initialisation can be found in figure 4.16. Both backward modelling algorithms are nearly independent of the size of the initialisation. The BACKMODELGIANT algorithms has some difficulties with very low initialisation in the case of ordered pattern presentation.

The results show that learning with virtual inputs is extremely powerful on symmetrical environments. Very small weight initialisation, which do not bias the network beforehand are handled graciously.

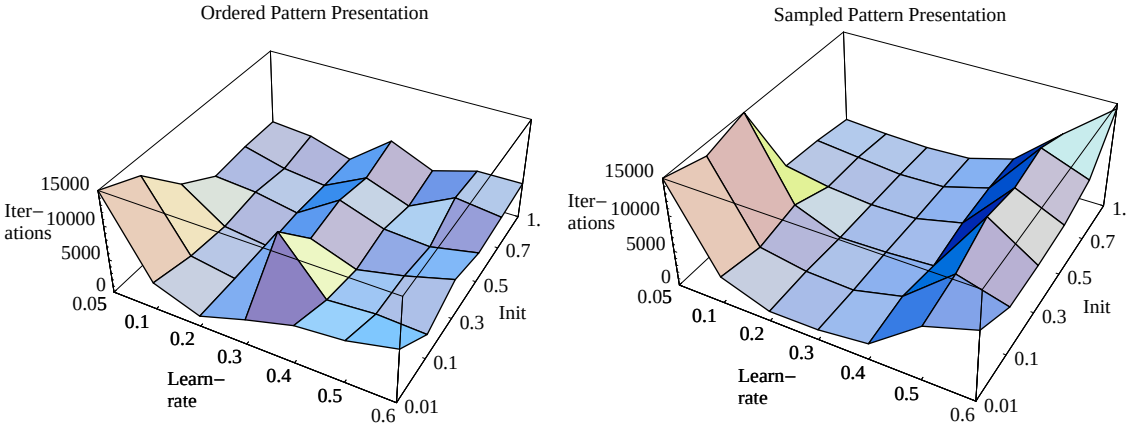

Figure 4.14: BackModel: Dependency on the input distribution for 4-Parity. The simulation was aborted after 32 000 iterations. *Left*: ordered pattern presentation; *Right*: patterns sampled. (see figures 4.10 and 4.12 for comparison)

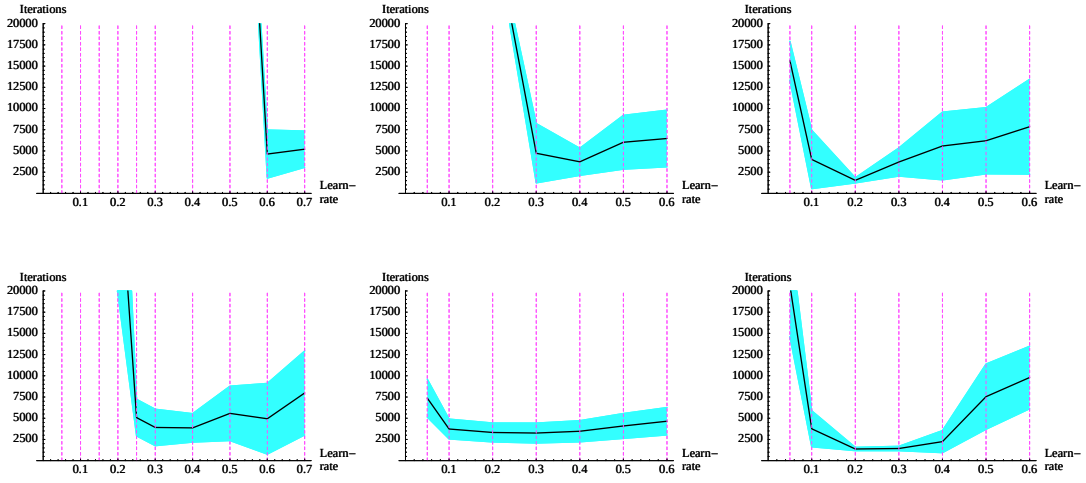

Figure 4.15: Comparison of learning rate dependency for 4-Parity with low weight initialisation ($[-0.01, 0.01]$). The cyan (dotted) vertical lines mark the used parameter value. The thickness of the blue (shaded) hose around the black line indicates the standard deviation. *Upper row*: ordered presentation; *Lower row*: sampled presentation. *Left*: Backprop; *Center*: BackModelGiant; *Right*: BackModel.

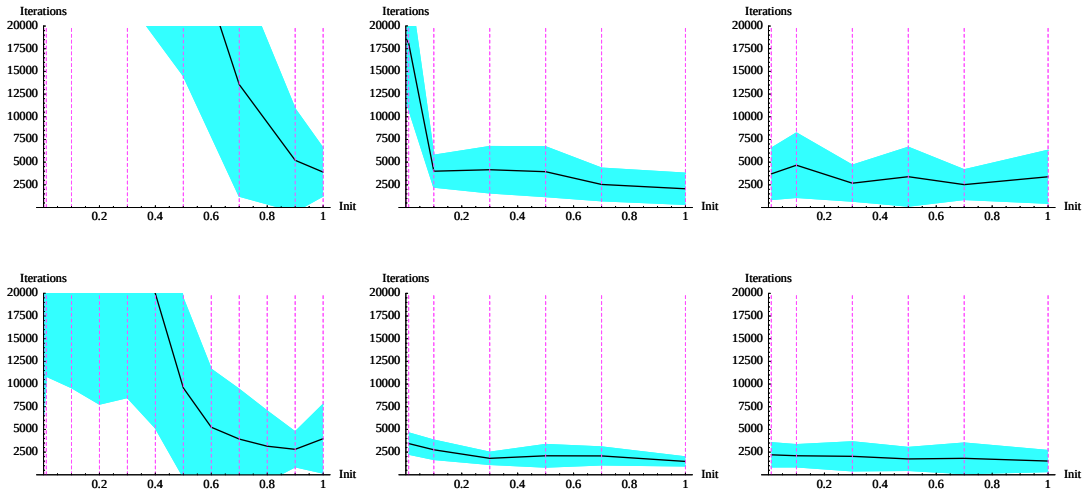

Figure 4.16: Comparison of weight initialisation dependence for 4-Parity with $0.1 \leq \eta \leq 0.4$. The cyan (dotted) vertical lines mark the used parameter value. The thickness of the blue (shaded) hose around the black line indicates the standard deviation. *Upper row*: ordered presentation; *Lower row*: sampled presentation. *Left*: BACKPROP; *Center*: BACKMODELGIANT; *Right*: BACKMODEL.

4.6.3 Encoder

The learning task of the Encoder problem is basically to produce the identity function, but with a bottleneck in the network. The input is a natural number from 0 to n encoded unary, i.e. one of n inputs is 1 and all others are 0 (or -1). The nominal output is the same, except of a scaling factor. The network has $h < n$ hidden units. For example if we set $h = \log_2 n$ the network has to "invent" something like binary encoding in the hidden layer.

An encoder-like schema occurs in the sensorimotor loop with a world model as discussed later in section 6. If we consider the prediction of sensors for the future, the most sensors do not change significantly within two time steps and therefore the input is equal to the output.

First of all, we want to validate that the layer-wise approach is really able to deal with deep networks. Therefore, we want to analyse the performance of the algorithms for multiple hidden layers. We will use the 10-5-10 encoder, which means 10 input units and 5 units in the hidden layers. This is not a particular difficult task in general. The network has all feed-forward connection between adjacent layers. The training process is the same as for XOR (see 4.6.2). Patterns will be presented randomly (sampled according to an equal distribution). The hidden and output units have a hyperbolic tangent activation function and a bias. Inputs and outputs are $\{-1, 1\}$ and $\{-0.9, 0.9\}$ respectively.

For one hidden layer the improvement through backward modelling is just marginal. Let us

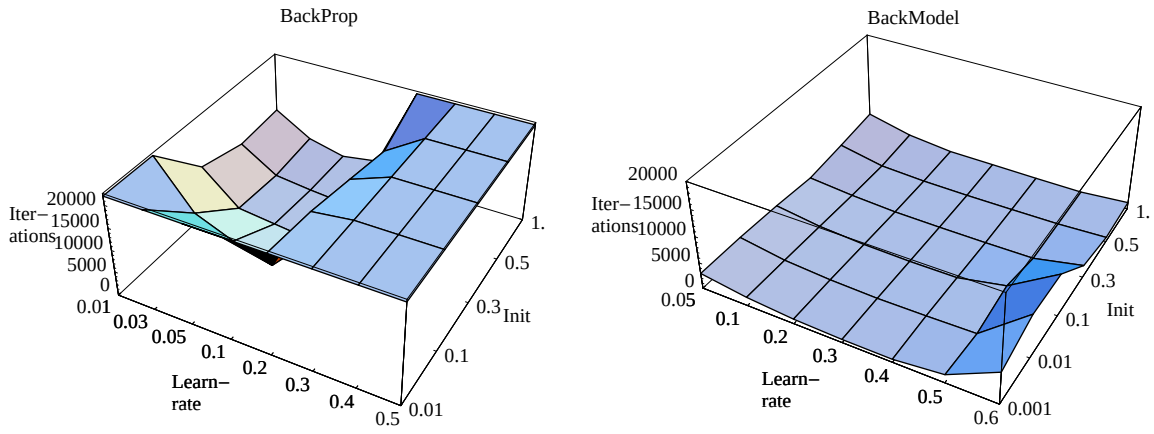

Figure 4.17: Encoder 10-5-10, 3 Hidden layers: Learning rate - Weight initialisation - Iterations. Parameters: maximal iterations: 22 000, $\gamma = 0.5$. *Left*: BACKPROP; *Right*: BACKMODEL.

consider a network with three hidden layers.

Three Hidden Layers

In figure 4.17 the performance of BACKPROP and BACKMODEL is displayed. Backpropagation needs a learning rate within a very small interval ($[0.03, 0.1]$) and a high initialisation as well. BACKMODEL is again tolerant in respect to both parameters and needs less iterations (pattern presentations) in general. Please note that after 22 000 iterations the learning process was aborted. Therefore, there are flat regions at this level which do not reflect the true behaviour.

In figure 4.18 the different algorithms are directly compared. One can clearly see the strong dependency of BACKPROP and BACKMODELGIANT on the learning rate and on the initialisation, where the latter algorithm is slightly better. For low weight initialisation the over-all algorithms perform very badly, whereas the layer-wise approach succeed very fast for a wide range of learning rates ($0.01 \leq \eta \leq 0.6$). For high weight initialisation all algorithms reach nearly the same performance for a learning rate around 0.05.

More Hidden Layers

In the last section we have seen that the layer-wise backward modelling algorithm outnumbers the "over-all" algorithms for three hidden layer networks. In this section we want to consider the performance dependent on the number of hidden layers. Figure 4.19 compares the learning speed of BACKPROP and BACKMODEL with local and over-all additional backpropagation learning step. Please be aware of the maximal number iterations of 44 000. It warps the line

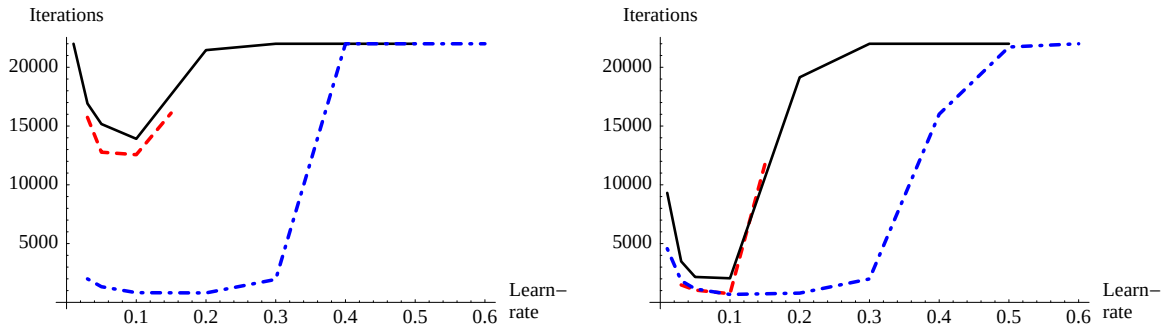

Figure 4.18: Encoder 10-5-10, 3 Hidden layers: Learning rate - Iterations. Black line (solid): BACKPROP; Red line (dashed): BACKMODELGIANT; Blue line (dot-dashed): BACKMODEL. Parameters: maximal iterations: 22 000, $\gamma = 0.5$. *Left*: low initialisation (≤ 0.1); *Right*: high initialisation (≥ 0.3).

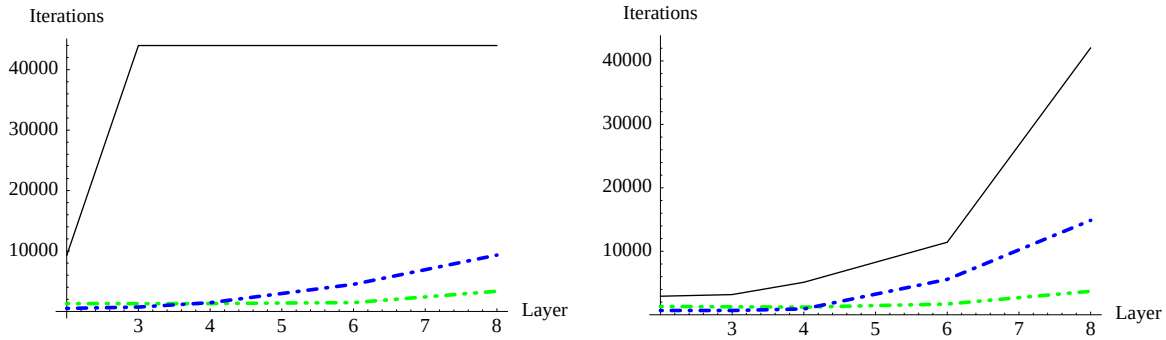

Figure 4.19: Encoder 10-5-10: Hidden layers - Iterations. Black line (solid): BACKPROP $\eta = 0.03$; Blue line (dot-dashed): BACKMODEL $\eta = 0.1$; Green line (dot-dot-dashed): BACKMODELFULLBP $\eta = 0.03$. Parameters: maximal iterations: 44 000, $\gamma = 0.5$. *Left*: low initialisation (≤ 0.1); *Right*: high initialisation (≥ 0.3).

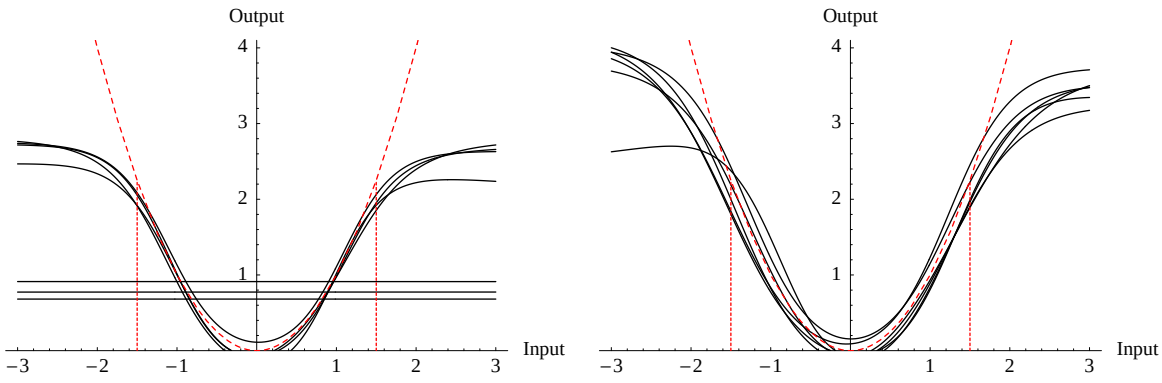

Figure 4.20: Square: Extrapolation, Network output after 6 000 pattern presentations. Parameters: $\eta = 0.05$, $\gamma = 0.5$. *Left*: BACKPROP; *Right*: BACKMODELFULLBP.

for BACKPROP significantly. For each algorithm the optimal values for the learning rate have been picked.

We have seen that the layer-wise backward modelling approach enabled learning of very deep networks with high speed nearly independent of the weight initialisation. The slow down through an additional hidden layer is comparable low.

4.6.4 Square

In order to verify the generalisation improvement through backward modelling we will use a parabolic regression problem. The input and output is one dimensional and conceivable simple. It follows the equation:

$$y = x^2 + \xi \ , \tag{4.31}$$

where ξ is noise. We will use a network with one input and one output unit, both have a linear activation function, and h-hidden units in one hidden layer with a hyperbolic tangent activation function. The network has all feed-forward connection between adjacent layers.

Extrapolation

The network used here has $h = 4$ hidden units. The training dataset involved samples in the interval $[-1.5, 1.5]$ with white noise. The test dataset includes 150 samples in the interval $[-3, 3]$. After 6 000 pattern presentations, the training with learning rate $\eta = 0.05$ was stopped. In figure 4.20 you can see randomly picked instances for weight initialisations 0.001, 0.1, 0.5, and 1.0. We can see that BACKPROP, even if trained successfully, differs very much from the

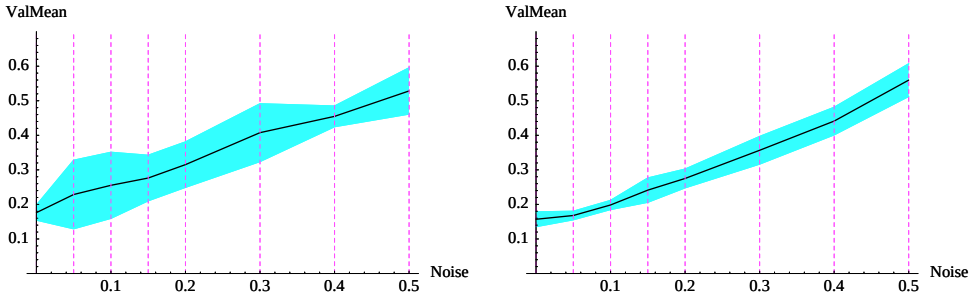

Figure 4.21: Square: Performance on noisy data. Validation mean error after 6 000 iterations. The cyan (dotted) vertical lines mark the used parameter value. The thickness of the blue (shaded) hose around the black line indicates the standard deviation. *Left*: BACKPROP for init $= 1.0$; *Right*: BACKMODELFULLBP.

target model outside the trained area. BACKMODELFULLBP extrapolates much better but tends to be a bit less accurate inside the trained interval. The reason for this symptomatic difference is the fact that the backward modelling approach keeps the hidden units away from the saturation region, see section 4.4.2.

Noise

In this section we want to examine the influence of noise. We use a clean dataset in the range $[-1.5, 1.5]$, where each epoch independent noise is added. The validation set has 40 samples in the same interval as the training set. One might expect that backward modelling performs badly here, because the response increasing in driven by noise and might cause the model to diverge. However, the experiments have shown that this effect is not significant or even measurable. Figure 4.21 shows the mean squared error on the validation set after 6 000 iterations in dependency on the strength of the noise. An error of about 0.6 means that the model was just a straight line, i.e. the task was not successfully learnt and a value of about 0.2 means a good approximation of the model. We see that noise is handled by both learning algorithms in nearly the same way.

4.6.5 Experiences

In this section we want to describe some further experiences we made through the development of the algorithms.

Because our first implementation of BACKMODELGIANT was quite slow, especially if the units have been in the saturation region, we tried the following relaxation of the units. If a unit

is activated above a certain threshold all incoming weights are decreased by a fixed factor. This should cause the unit to stay away from total saturation without limiting the weights to some range. Our practical results with BACKPROP have shown that this technique does not help the learning process, instead it disturbs it. At first glance it sounds strange because whether a unit (with tanh activation function) has an activation of 0.97 or 1.0 should not make any significant difference to the forthcoming network. This argument is only correct if one considers one particular pattern. Let us consider a certain unit u and denote the pattern which causes u to be activated above the threshold *strong patterns*, whereas *weak patterns* activate u in the range of high response somewhere between -0.5 and 0.5. A decrease of all input connections of u, even by a small portion, will already change the activation of u for a weak pattern significantly. Therefore, the relaxation effects the network so that it performs badly on weak patterns, which might have been perfectly represented before. One can think of another reason as well. Unit u can be considered as a constant for the strong patterns. That means small weight changes induced by other patterns do not effect the performance on strong patterns. What we learn from that is, that there are reasonable representations that include very high saturated units for some areas of the regular input space. Obviously there are other representations, which avoid that. These representations are preferred by backward modelling algorithms, but which may need a larger network.

4.7 Summary

We have seen, that backward modelling is a general approach to machine learning, which is based on the inversion of the learning system. Through that, virtual inputs are generated, which can be used for learning. The theoretical analysis showed basic effect like response increasing and symmetry breaking. Aside, we demonstrated that backward modelling can be nicely applied to neural networks and discussed three learning algorithms: BACKMODELGIANT, BACKMODELSIM, and BACKMODEL. The first two are algorithms in the spirit of backpropagation, which consider the network as a whole. BACKMODELSIM turned out to be unsuccessful. The last one is a layer-wise algorithm, that decomposes the learning into learning on small subnetworks. We have investigated three variations, which differ in the way how the additional backpropagation learning step is performed. The combination of layer-wise backward modelling and over-all backpropagation (BACKMODELFULLBP) has shown to be theoretically and practically superior.

We supported the theoretical results with practical experiments and achieved remarkable improvements, even on neural network specific problems. All experiments showed the nearly complete independence of the size of the weight initialisation. This is a great success, which is especially important for the use of homeokinesis [61]. That is an approach to self-organised

autonomous robot control, where backward modelling seems to be perfectly suitable. Dependency on weight initialisation is also a major problem of the back percolation algorithm (see 3.3.4), which follows similar ideas of network inversion. However, it does not use virtual data points and such like, and it is specialised to neural networks.

Spontaneous symmetry breaking and response increasing was discussed at experiments with XOR and n-Parity, which confirmed the theoretical predictions. The Encoder problem was used to justify the capability of backward modelling to train deep networks, i.e. networks with many hidden layers. This is a known problem of backpropagation and many derivates. The results show a remarkable performance even on networks with more than four hidden layers using the layer-wise algorithm.

Backward modelling ballances the response of the hidden layers of a neural network to be in the most operable range of the unit's activation function. This causes responsive models, which is desirable for robot control. Apart from that, it prevents the units becoming completely saturated, which causes a generalisation improvement. This assertion was supported by experiments on the nonlinear regression problem Square. We could also show that backward modelling is still effective on noisy environments.

The property to respect the current model while learning has not been investigated. However, because learning is performed with data points more close to the current model it seems likely that backward modelling shows some kind of hysteresis in respect to large model changes.

Through the development of the algorithms the biological constraints of local computation has been respected. Nevertheless, the algorithms suffer from the same biological implausibility as the standard backpropagation algorithm. The transformation into a biologically more feasible version along the line of GeneRec [63], has not been elaborated, but seems likely to be possible.

The proposed algorithms have a higher but still moderate computational complexity in comparison to backpropagation. More precisely, they have the same complexity class and introduce a factor 10 in the average case.

Chapter 5

Active Learning using Backward Modelling

This section describes how backward modelling can be used in the realm of active learning. In section 2 different approaches to active learning have been summarised. Let us now see how the approach proposed here differs and how it can be classified.

The main idea of active learning is to perform some kind of action on the environment in order to manipulate the patterns that are presented to the network. The goal is to get more informative patterns to speed up learning. For that, some kind of information gain measure is required. In the case of backward modelling we will use the shifts and the virtual inputs as measures. There are two possible ways. Either one uses the size of the shifts as a measure of information gain or one uses the virtual inputs as desired network inputs, which forms a concrete query to the environment. The first one falls into the range of pattern selection algorithms, whereas the second one is a query algorithm.

5.1 Shifts as Information Gain Measure

As discussed in section 2.3 there are different heuristic approaches to estimate the information gain. Many of them use the size of the error value [11; 13; 14] and some the response strength [17]. We suggest to use the size of the shift instead. Patterns which produce a high error value or patterns where the response of the network is low will be preferred, because the shift will be large. This is coherent with using the error value as measure for the information gain, but contradicts with the approach of Engelbrecht [17] to prefer patterns where the response of the network is high. The reasons why we propose the opposite strategy are as follows: The network learns fast in regions of high response anyway, so we do not need to present particularly much

patterns there. Regions of slow response mean either saturation or flat linear behaviour. Both are areas where the representation is likely to be bad and the backward modelling technique is particular appropriate for these cases. Please note, that for marginal errors at the output the shifts are also not big, i.e. if the model is in a saturation region but correct there, our interest is low as well.

Having this measure of estimated information gain, one can use the same techniques for selection, growing, or pruning as with other measures. We will only consider epoch-wise pattern selection using the idea of large-next-day and small-next-day subsets [15]. In practice we calculate a probability for each pattern in the training set according to the size of the shifts. To ensure every pattern will be presented eventually we introduce a minimal probability, which is a fraction of the probability of the most likely pattern. In the following we will refer to this strategy with the name PROPSELECTION m, where m is the fraction of the minimal probability. For examples a value of $m = 1$ implies an equal distribution because the minimal probability is the same as the maximal probability, whereas a value of 0.1 means that every sample has at least one tenth of the probability of the most frequent pattern.

5.2 Virtual Inputs as Queries

Let us now consider a completely different approach to active learning. We will request or query for new inputs on the fly. That means, after each training pattern the network requests a new pattern. This stands in contrast to the pattern selection technique, where global selection after many pattern presentions is performed. We use virtual inputs produced by the backward modelling technique as direct queries. This may lead to new input patterns, which do not belong to the training set and cause the environment to be explored.

Selection methods can be used to speed up learning on a fixed dataset whereas this query method is more appropriate for real world communication. However, it can also be used in the classical learning schema with the use of some algorithms for finding the next pattern from the environment. First of all, we need to find a pattern which should be considered as the requested one, and second there should be some strategy when to use the requested pattern and when a regular one.

5.2.1 Nearest Pattern Selection

The following methods can be used to determine the pattern from the fixed dataset which should be used instead of the queried network inputs:

- Lowest Euclidean distance.

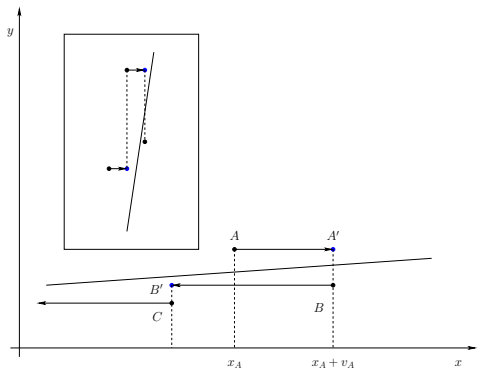

Figure 5.1: Backward modelling and active learning. A data point A at x_A is presented. After the usual BACKPROP learning step the virtual input at $x_A + v_A$ is send to the teacher as a query, where the teacher provides the data point B. This is shifted to towards the regression line and B' is the virtual data point, with C as the corresponding real data point. The range of exploration strongly depends on the steepness of the regression line (response strength of the model), see the small upper insert with a much steeper line.

- Lowest Euclidean distance, but previous pattern is moved away.
- Pick a pattern according to a Boltzmann distribution with a certain temperature Θ:

$$p_i = \frac{e^{-\Theta * d_i}}{\sum_j e^{-\Theta * d_j}} \tag{5.1}$$

  where d_i is the distance and p_i is the probably of pattern i.

Let us now consider a continuous environment, where patterns can be requested freely. The active learner will scan the environment dependent on its own state and on the structure of the environment. Figure 5.1 shows that active learning with virtual inputs will sample the environment in a large range if the response is small or the error is large. In contast, it will sample the environment in a very small range if either the response is high or the error is tiny. In case of a small error another force is needed to bring the learner to different areas of the environment. Otherwise the learning will only focus on a very small area of the environment. Therefore, one needs to swap between active pattern querying and regular pattern presentation.

5.2.2 Presentation Strategy

Possible strategies for switching between queried pattern and regular pattern are:

- Use the queried pattern as long as it is different from the currently presented one (UNEQUAL). This strategy is just suitable for discrete environments, i.e. learning with a fixed dataset.

- Use the queried pattern if it is far away from the currently presented one (FAR), i.e. if the Euclidian distance between the two is larger than a threshold ρ.
- Both methods can be combined with a limitation k of how many active pattern request are allowed in a row. For instance FARLIMIT.
- Use a weighted average between the queried pattern and the regular pattern (MIDDLE). The weight factor fill be denoted as α. This strategy seems to be more reasonable for applications like robot control, see below.

The main idea is that the environment will be scanned locally if it is not understood. Unfortunately noise attracts the focus of the active learner as well, which is not quite desired.

5.3 Results

The approach to use virtual inputs as queries will be used in the next chapter for robot control, where continuous environments are used. For a fixed and discrete environment it can be classified as pedagogical pattern selection at a small scale and did not improve learning as far as we could experience.

In order to assess the idea of pattern selection with shifts we performed experiments on different problems. On the Encoder problem we could not record any significant improvement in the learning speed in comparison to passive learning. The reasons might lie in the structure of the problem itself. Since the Encoder problem is basically a classification problem with one sample per class we cannot effort to omit a few samples for a longer time. The same holds for n-Parity. Therefore we investigate the Square problem as explained in section 4.6.4 on page 60.

5.3.1 Square

The Square problem was already specified in section 4.6.4. We use the same network with 4 hidden units but a dataset with few training samples and a wider validation set. More precisely, the training dataset involves 10 samples in the interval $[-1.5, 1.5]$ with little white noise $\xi \in (-0.15, 0.15)$, and the validation dataset involves 20 samples in the interval $[-2, 2]$. This is used because the generalisation loss as mentioned in section 4.6.1 should be used as stopping criteria.

PROPSELECTION m (see section 5.1) is used as pattern selection strategy. In figure 5.2 and 5.3 the mean test set error and the learning speed are shown for termination with $\mathrm{GL}^{10}_{0.05}$. Let us recall the influence of m on the pattern selection. A value of 1 means no active learning at all,

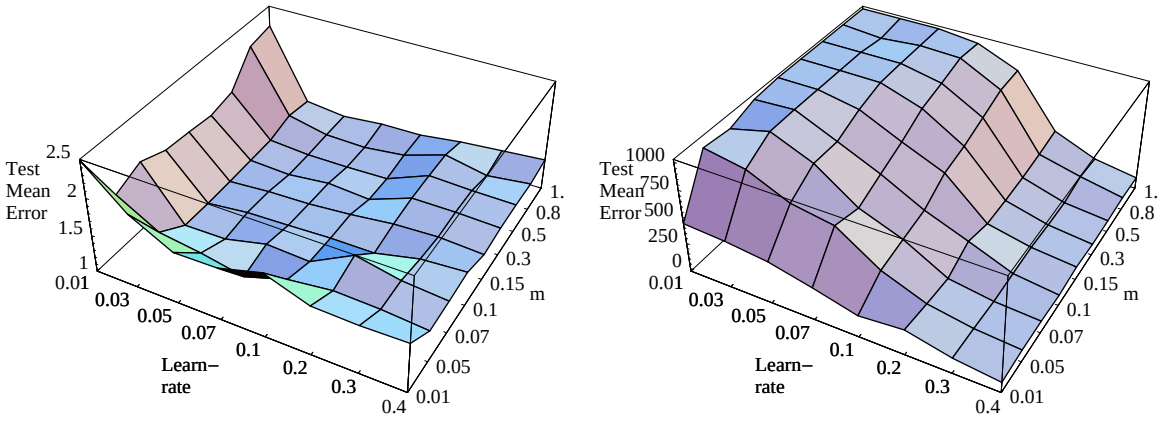

Figure 5.2: Active learning with BACKMODELFULLBP using shifts as informativeness measure. Pattern selection with PROPSELECTION m. Termination $\mathrm{GL}^{10}_{0.05}$. The smaller m the more patterns with large shift are presented. Parameters: $\gamma = 0.5$. The cyan (dotted) vertical lines mark the used parameter value. The thickness of the blue (shaded) hose around the black line indicates the standard deviation. *Left*: Learning rate - m - Mean test set error; *Right*: Learning rate - m - Epochs.

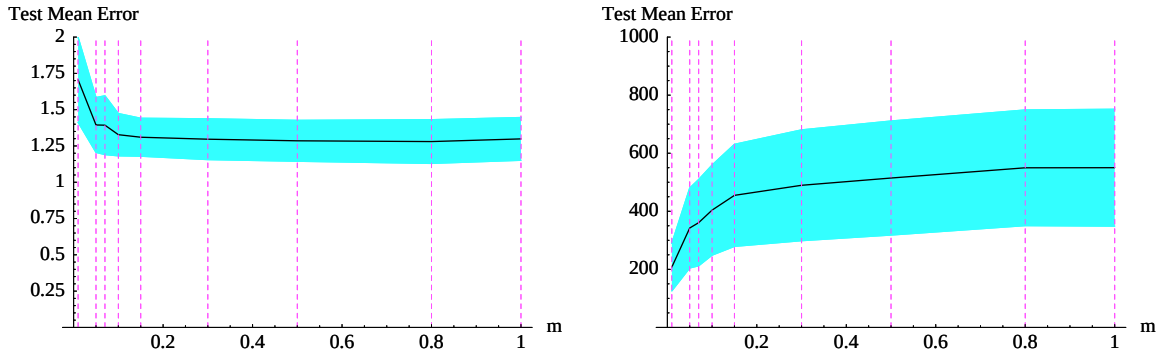

Figure 5.3: Active learning with BACKMODELFULLBP using shifts as informativeness measure. Pattern selection with PROPSELECTION m. Termination $\mathrm{GL}^{10}_{0.05}$. *Left*: m - Mean test set error; *Right*: m - Epochs. The cyan (dotted) vertical lines mark the used parameter value. The thickness of the blue (shaded) hose around the black line indicates the standard deviation.

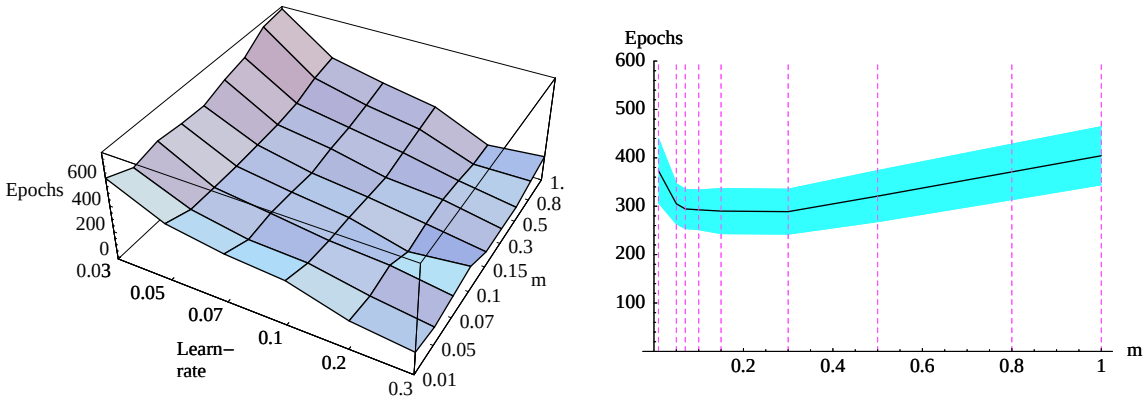

Figure 5.4: Active learning with BACKMODELFULLBP using shifts as informativeness measure. Pattern selection with PROPSELECTION m. Fixed threshold termination (0.15). *Left*: Learning rate - m - Epochs. *Right*: m - Epochs for low learningrates.

and a small value of m means that only patterns with a large shift are selected. We see that active learning decreases the learning time significantly, but increases the mean test set error if m is chosen too small. In order to analyse the influence of active pattern selection on the learning time more precisely, we use a fixed threshold termination. The training is stopped if the squared summed error on the validation set was below a certain threshold. That means that we can ensure a good representation. In figure 5.4 the learning speed in dependency on the learning rate and m is plotted. We can see that active pattern selection can reduce the training time. However, if m is chosen too small the training time can even increase. This can be explained by the fact that only a small subset of all patterns are presented. In the three dimensional plot (left side of figure 5.4) one can see that active pattern selection is especially helpful for low learning rates.

Chapter 6

Application to the Sensorimotor Loop

6.1 Introduction

The sensorimotor loop is the name of the circular information flow occurring to a mobile agent. The agent receives information from the environment with its sensors and produces some actions with its motors or actuators. This has an impact on the environment seen by the agent. Figure 6.1 shows a simple robot with two wheels, a number of infra-red sensors and a controller in the sensorimotor loop.

Following the ideas of homeokinesis we introduce an adaptive self model or world model, which tries to predict the results of the actions of the robot in terms of changes of the sensor values. The inputs for the world model are the controller and sensor values. The world model is learnt with supervised learning to predict the sensor values for the next time step.

A central problem in robotics is the concomitant learning of model and controller, known as the cognitive bootstapping problem. In order to gather knowledge about the world, the robot must explore the world appropriately. For that, the robot has to *know* how to choose convenient

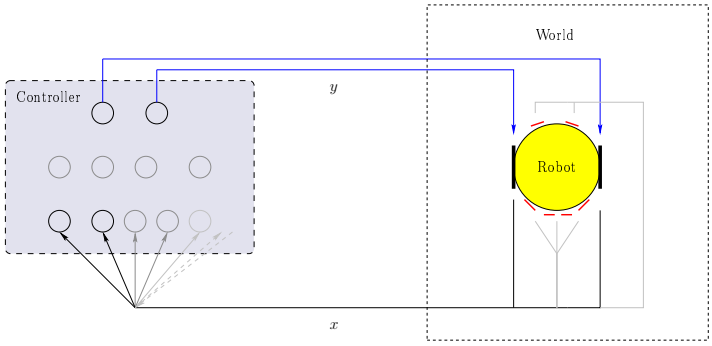

Figure 6.1: Simple robot in the sensorimotor loop. x sensor values, y motor values.

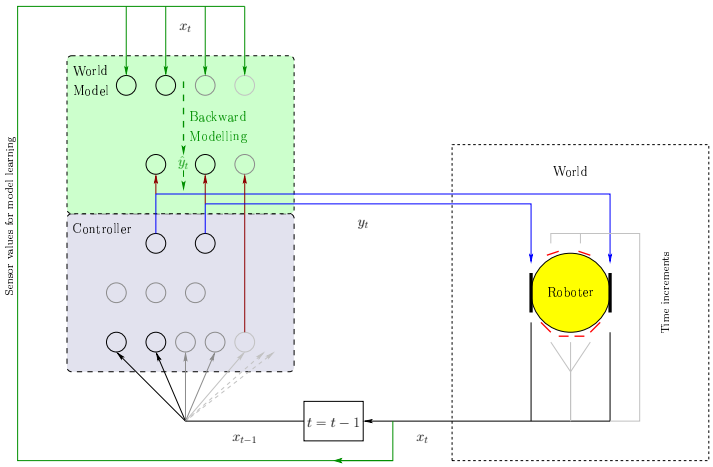

Figure 6.2: Sensorimotor loop with world model and backward modelling. Please note that the time is incremented in the world, therefore we start the time line at the robot and obtain the sensor values x_t at time step t. The controller receives the delayed sensor values x_{t-1} from the previous time step, whereas the nominal output for the world model is given by x_t. The controller values are denoted as y_t. $\hat{y}$ is the virtual input of the model which is equal to *virtual controller values*.

actions. The bootstrapping problem arises, if the knowledge about the corresponding sensorimotor coordination is to be aquired on the basis of an adaptive world model both of which start from *tabula rasa* initial conditions.

Let us consider the used scenario as sketched in figure 6.2. The main difference to the simple sensorimotor loop (figure 6.1) is the adaptive world model, which is introduced here. In order to utilise the world model, one needs to propagate the learning signal through the world model to the controller. This causes the controller and the model to learn simultaneously, and the controller is forced to produce actions with predictable outcomes. We now use backward modelling in the world model. As stated above, the input to the world model are the controller and sensor values. Backward modelling produces a shift at these inputs, which means that we obtain *virtual controller values* and *virtual sensor values*. As the name suggests, the virtual controller values can be sent to the robot. However, in order to produce virtual controller values for time step t one needs sensor values for time step $t + 1$. Since we do not have these values before sending actuator commands, we let the controller live in the past. In other words the sensor values received by the controller are delayed. This introduces a time loop error which is discussed by Der [64]. In conclusion, in each time step we learn the world model and the controller using the error between the predicted sensor values and the actual sensor values. Additionally we obtain virtual controller values with backward modelling. We can use the strategies described in section 5.2.2 to obtain the actual actuator commands from the regular controller values and the virtual controller values. We obtain a dynamics between controller and model learning, with emerging behaviour for environments of moderate complexity.

In this work the learning of the controller shall not be considered. Instead we analyse the influence of a restricted controller to the adaptive world model. In the following section we want to take a look at an example and investigate the use of active learning to produce proper world models.

6.2 Turni

We will now consider a simple example with a Khepera[1] like robot. The sensors are the measured wheel velocity from the left and the right wheel, and actuators are the two motor values for the wheels. The controller can be written as $y = K(x)$, where $y, x \in \mathbb{R}^2$. The dynamics read:

$$x_{t+1} = \mathcal{A} y_t + \xi \qquad \xi \text{ noise} \tag{6.1}$$

$$\mathcal{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathcal{I} \qquad \mathcal{I} \text{ unit matrix} \tag{6.2}$$

The model has to find the mapping between the controller values sent to the motors and the measured velocity values. We will use a linear model which can be described as:

$$x_{t+1} = \mathcal{W} \cdot y_t \ , \tag{6.3}$$

where the 2×2 matrix $\mathcal{W}$ should be learnt. The task seems to be quite trivial, but we will see in the following that this is not always the case.

Let the controller be restricted in the sense that it only allows the robot to rotate in place with periodic turning speed. More precisely $y_1 = -y_2 = \sin(\omega t)$. The controller lives in the subspace spanned by the vector $\binom{1}{-1}$. First of all the training of $\mathcal{W}$ should be analysed. For that we consider the usual error

$$E = \|x - \mathcal{W} y\| = \|\mathcal{A} y + \xi + \mathcal{W} y\| = \|(\mathcal{I} - \mathcal{W}) y + \xi\| \ . \tag{6.4}$$

The scalar product is used as the norm. Recalling that $\overline{\xi} = 0$, we obtain

$$E = (\mathcal{I} - \mathcal{W}) y)^2 + \xi^2 \ . \tag{6.5}$$

[1] Widely known and used two wheel robot designed and distributed by K-Team Corporation, see `http://www.k-team.com/robots/khepera`

The update rule for $\mathcal{W}$ reads

$$\begin{aligned}\Delta\mathcal{W} &= \frac{\partial}{\partial\mathcal{W}}E = (\mathcal{I}-\mathcal{W})y\cdot -y^T \\ &= \mathcal{W}\mathcal{Y}-\mathcal{Y} \qquad\qquad \text{with } \mathcal{Y}=y\cdot y^T\ .\end{aligned} \tag{6.6}$$

The training converges if $\Delta\mathcal{W}=0$. In our case y is a multiple of $\binom{1}{-1}$, nevertheless we use $y=\binom{1}{-1}$ for simplicity and get

$$\mathcal{Y}=\begin{pmatrix}1 & -1\\ -1 & 1\end{pmatrix}\ . \tag{6.7}$$

Since $\mathcal{Y}$ is singular the equation $0=\mathcal{W}\mathcal{Y}-\mathcal{Y}$ can not be solved for $\mathcal{W}$. By construction $\mathcal{Y}$ is a projector satisfying $\mathcal{Y}\mathcal{Y}=\alpha\mathcal{Y}$, with $\alpha=\frac{1}{2}$ in our case. Without loss of generality, we can write

$$\mathcal{W}=\alpha\begin{pmatrix}1 & -1\\ -1 & 1\end{pmatrix}+\mathcal{Q} \qquad \text{with } \mathcal{Q}\,\mathcal{Y}=0 \tag{6.8}$$

$$\mathcal{Q}=\begin{pmatrix}a & a\\ b & b\end{pmatrix} \tag{6.9}$$

It can easily be shown that equation (6.8) holds for $y=k\binom{1}{-1},\ k\in\mathbb{R}$. If a small damping of the matrix elements – which are network weights – is used, $\mathcal{Q}$ will become $\underline{0}$. This means $\mathcal{W}$ will converge to the degenerated matrix

$$\mathcal{W}_{deg}=\frac{1}{2}\begin{pmatrix}1 & -1\\ -1 & 1\end{pmatrix} \tag{6.10}$$

instead of $\mathcal{W}=\mathcal{A}$. The world model would not concur with the real world. Devising a new control strategy is not possible on the basis of this world model so the robot is then trapped in the degenerated rotational mode of behaviour.

Active learning in conjunction with backward modelling is able to solve this problem. As described above the abserved sensor values are used to produce virtual controller values, which are the real controller values shifted by v. What does v looks like? It is obtained from the inversion of the model.

$$v=\mathcal{W}^{-1}\xi \qquad\qquad \xi \text{ modelling error.} \tag{6.11}$$

$\mathcal{W}_{deg}$ is singular and cannot be inverted, but in practice we will rather have something like

$$\mathcal{W} = \frac{1}{2}\begin{pmatrix} 1 & -1+d_1 \\ -1+d_2 & 1 \end{pmatrix} \qquad |d_i| \ll 1$$

$$\mathcal{W}^{-1} = \frac{1}{2(d_1+d_2+d_1d_2)}\begin{pmatrix} 1 & 1-d_2 \\ 1-d_1 & 1 \end{pmatrix} \approx \frac{1}{2(d_1+d_2)}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Let us consider the shift for $\xi = \binom{1}{1}$

$$v = \mathcal{W}^{-1}\binom{1}{1} \approx \frac{1}{d_1+d_2}\binom{1}{1}$$

which is very large for small d_i. This means that any component of ξ in direction of $\binom{1}{1}$ – which means *go straight* – produces a large shift. When using the resulting virtual controller values to steer the robot, it will move forward or backward and the new sensor values will have a strong component in the direction $\binom{1}{1}$ as well. When training the model with the new sensor values, we realise that the error, and therefore the learning signal, is large as well; because $\mathcal{W}$ is just projecting into the subspace $\binom{1}{-1}$. Consequently, the model is trained to reflect the true world and to project to the full sensor space quickly. On that basis new control strategies could be contrived. However, over time the model will degenerate again and at some point active learning again takes effect.

6.2.1 Simulation

Before we look at the results of the experiments, let us discuss the impact of the pattern presentation strategies as mentioned in section 5.2.2. For continuous environments there are just two basic variants: MIDDLE and FAR strategy where the latter can be combined with a LIMIT. The MIDDLE strategy uses a weighted arithmetic average of the real controller output and the virtual controller output as actuator commands.

$$y_\alpha = \alpha \cdot y_K + (1-\alpha) \cdot \hat{y}_{model} \ . \tag{6.12}$$

Typical values for the weight α lie between 0.05 and 0.5.

What behaviour do we expect? The robot will rotate and if the model is degenerated enough the robot will start to go forward or backward. We expect an equilibrium between the strength of the *go straight* component and the model degeneracy.

The second strategy is FAR, which switches between active learning control and regular controller based control. Active learning is enabled if the shifts exceed a certain threshold ρ. If a LIMIT k is used, active learning is disabled after at least k cycles of active learning con-

trol. The expected behaviour is that the robot rotates in place until the model is degenerated enough and then the robot will nearly go straight forward or backward for a few cycles until the model is more or less correct for driving straight. After that the controller steers the robot again and causes it to rotate in place and the game starts again. The limitation of the active learning period makes sense, because the active behaviour can rapidly bring the robot into a complete different actuator space and in case the environment is complex it might not be able to model it well enough to decrease the shift below the threshold. Unfortunately, the threshold depends strongly on the network and the application. Reasonable values for the limit k depend on the sensor sampling time scale but a value of about 10 might be a good start.

Network

For the world model a linear feed-forward network with 2 input and 2 output units is used. The weights have a small damping term in order to let the robot forget things, which are not constantly proven. The damping will accelerate the model degeneration and the effect can be seen more clearly. The weight matrix of the network represents the matrix $\mathcal{W}$ from equation 6.3.

Behaviour

The simulation of the robot is very simple, see equation 6.1. The motor commands are just combined with additive white or coloured noise and are returned as wheel sensor values. In figure 6.3 the behaviour of the robot for passive and active learning is plotted. We consider the determinant of the weight matrix as a measure of the quality of the world model. If the determinant is zero the world model is degenerated to $\mathcal{W}_{deg}$, and if it is 1 the world model is equal to $\mathcal{A}$. For passive learning the robot is steered by learning the controller only and the forward-speed is zero. In the case BACKPROP is used for learning, the world model is bad, which can be seen by the extremely small determinant of the weight matrix. Backward modelling already does a better job in the passive learning schema, because it keeps the response of the model high. Active learning with virtual data points leads to much better representations of the world. The exact behaviour depends on the pattern presentation strategy and the parameters used. If we use the MIDDLE strategy active learning will influence the behaviour the whole time. In the bottom left chart of figure 6.3 one can see that the robot is nearly always going either forward or backward, however, the representation stays good (high determinant). The FARLIMIT strategy produces a different behaviour. Active learning is only activated for a short time. The overall behaviour of the robot is more closed to the passive case but the representation of the world is still good. We now take a closer look at

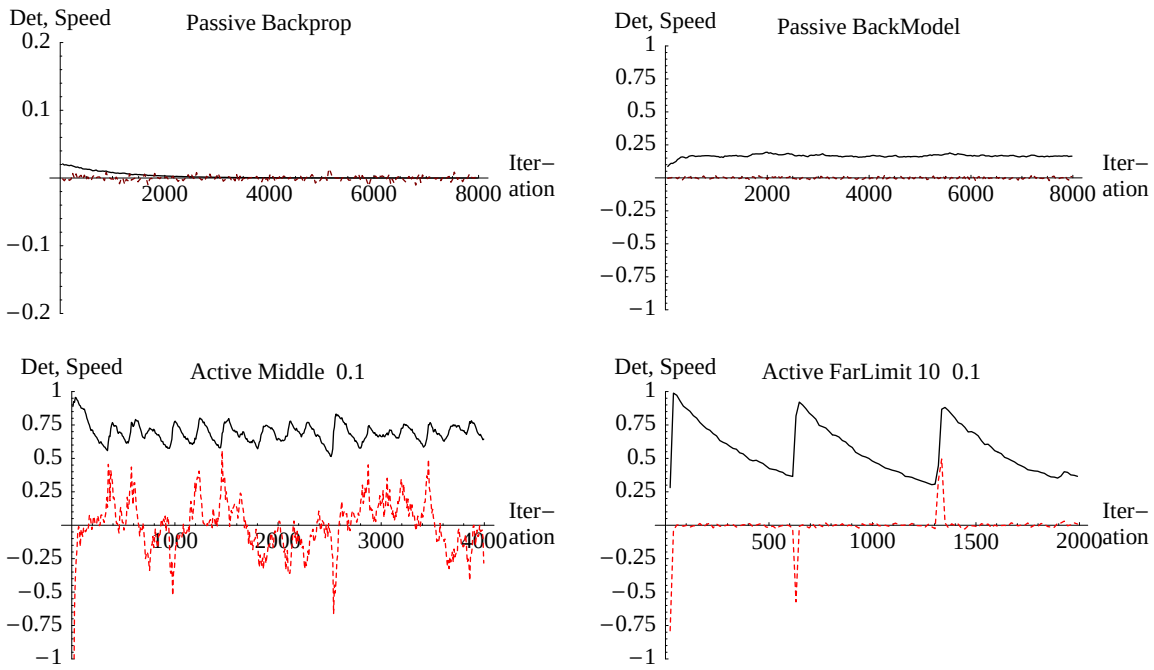

Figure 6.3: Model and behaviour of the Turni robot. The black line (solid) is the determinant of the matrix $\mathcal{W}$. A value of 1 means $\mathcal{W} = \mathcal{A}$ and a value of 0 means $\mathcal{W} = \mathcal{W}_{deg}$. The red line (dashed) is the forward/backward speed of the robot. Parameters: damping value: 0.001.
Top left: Passive learning BACKPROP; *Top right*: Passive learning BACKMODEL; *Bottom left*: Active learning with BACKMODEL and MIDDLE strategy with $\alpha = 0.1$: Determinant is high, but the robot is mainly going forward/backward;
Bottom right: Active learning with BACKMODEL and FARLIMIT strategy with $k = 10$ and $\rho = 0.1$: short periods of active behaviour.

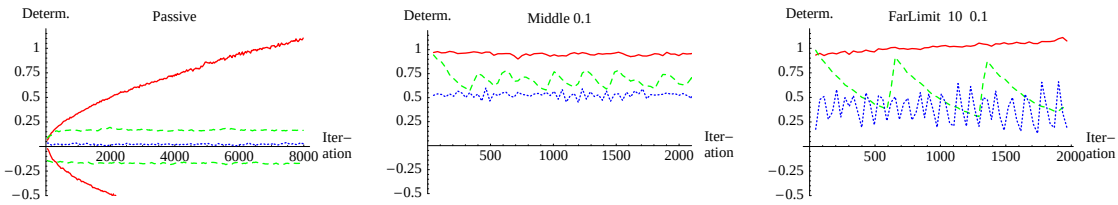

Figure 6.4: Determinant of the world model for Learning with BACKMODEL. A value of 1 means $\mathcal{W} = \mathcal{A}$ and a value of 0 means $\mathcal{W} = \mathcal{W}_{deg}$. The red line (solid) stands for no damping, the green line (dashed) is low damping (0.001) and the blue line (dotted) is high damping. *Left*: Passive learning; *Center*: MIDDLE strategy with $\alpha = 0.1$; *Right*: FARLIMIT strategy with $k = 10$ and $\rho = 0.1$.

the influence of the parameters.

Damping: In figure 6.4 the results for different values of the weight damping term are displayed. The passive case with backward modelling is already surprising. In the diagram there are two different instances for no and low damping, one with positive and one with negative determinant. This means, the model is trained to either something like $W = k \cdot \begin{pmatrix} d & -1 \\ -1 & d \end{pmatrix}$ or $W = k \cdot \begin{pmatrix} 1 & -d \\ -d & 1 \end{pmatrix}$, where $0 \leq d \ll 1$. The reason can be found in noise driven response increase of backward modelling. In the case of low damping the determinant is smaller and if the damping is too high the model degenerates completely. Active learning generates perfect models for no damping and reasonable models even for high damping. The MIDDLE strategy performs more constantly than FARLIMIT strategy, although it changes the behaviour of the robot more heavily (see figure 6.3).

MIDDLE Strategy: The parameter α influences the behaviour of the robot significantly. For low values the virtual controller values overwhelm the regular controller. For high values of α active learning will have no influence on the behaviour of the robot, but the world model cannot be improved either. In the example considered here a value of $\alpha \in [0.1, 0.3]$ can be recommended.

FAR Strategy: The selection of the threshold ρ is dependent on the architecture, the type of units, and the range of input values. In figure 6.5 the dependency of the behaviour on ρ can be seen. For a low threshold the behaviour is similar to the one of MIDDLE and if the threshold is chosen too high, active learning will almost never be activated. This encourages the investigation of an adaptive threshold. Recall the purpose of the distance threshold ρ and the limit k:

- Active learning should only be activated if shifts of significant size are produced.

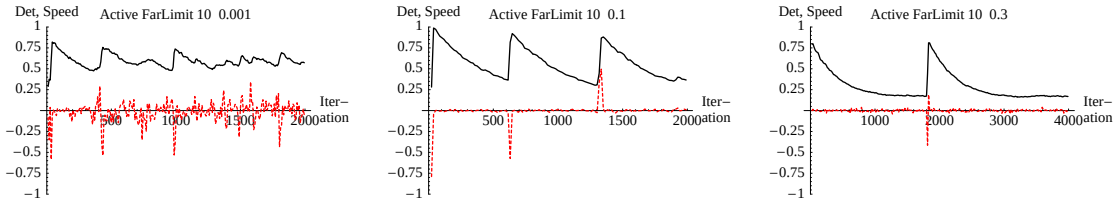

Figure 6.5: Behaviour and model of Turni for FARLIMITSTRATEGY with different thresholds. The cyan (dotted) vertical lines mark the used parameter value. The thickness of the blue (shaded) hose around the black line indicates the standard deviation. *Left*: $\rho = 0.001$; *Center*: $\rho = 0.001$; *Right*: $\rho = 0.001$.

- Most of the time the controller should dominate the behaviour.
- Periods of active learning should be limited to avoid endless active learning controlled behaviour.

Let us consider the FAR strategy with a distance threshold that is updated every time step with the formula

$$\rho_{t+1} = (1 - \beta)\, \rho_t + \beta \, \|v_t\| \ , \tag{6.13}$$

where β is the adaptation rate, which is to be chosen conveniently. Let us assume for now, that the shifts have roughly the same size. Over time the threshold will have the value of the average shift size. If the model is less correct now than before, a larger shift is produced, which will exceed the threshold and active learning will be activated. Considering the case that the shifts raise for a while – because the model is wrong at the new area of the sensor space – the threshold raises as well and at some point it will disable active learning again, which can be compared with the LIMIT. The obtained strategy will be called FARADAPT and satisfies the aims mentioned above.

FARADAPT Strategy: In figure 6.6 the behaviour of the robot and the adaptation of the threshold can be seen for low and high adaptation rates. One can see, that the choice of the adaptation rate is fairly uncritical, however, it needs to be chosen conveniently for the application in order to obtain the desired behaviour. Slow adaptation implies possibly long active learning phases, whereas fast adaptation guarantees frequent switching between active and passive phases.

Finally, we take a look at the opposite behaviour, going straight all the time. The controller

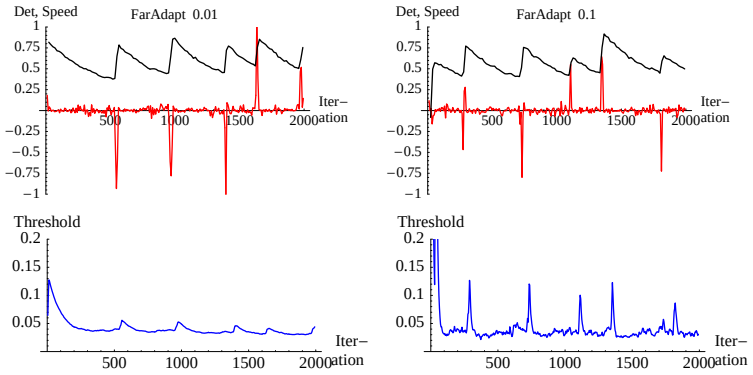

Figure 6.6: Behaviour and model of Turni for FARADAPT strategy. Top row: Model (determinant) and speed of the robot. The cyan (dotted) vertical lines mark the used parameter value. The thickness of the blue (shaded) hose around the black line indicates the standard deviation. *Bottom row*: Threshold adaptation during learning; *Left*: Slow adaptation $\beta = 0.01$; *Right*: Fast adaptation $\beta = 0.1$.

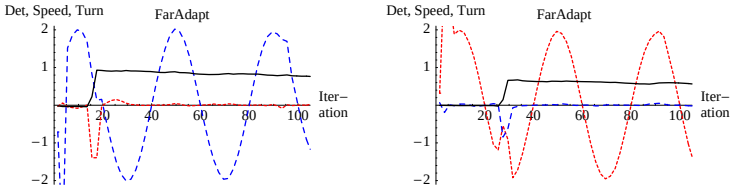

Figure 6.7: Full behaviour of the robot for FARADAPT strategy. Black line (solid): determinant of weight matrix; Red line (dotted): forward/backward speed; Blue line (dashed): turning speed. *Left*: Turni controller; *Right*: Straight controller.

is now $y1 = y2 = \sin(\omega t)$ and the model will converge to

$$\mathcal{W}_{deg} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} . \tag{6.14}$$

The evolving effects are analogue. In figure 6.7 both cases are displayed. The diagrams show the forward speed, the turning speed, and the determinant of the weight matrix in dependency on the time. One can nicely see how the regular behaviour induced by the controller is shortly overwhelmed by active learning in order to correct the world model.

6.3 Summary

The cognitive bootstrapping problem – which consists of the simultaneous training of the controller and world model starting from zero knownledge – is one of the central problems in robotics.

In order to devise new control strategies, the world model should not be degenerated. That means the world model should be able to represent the full behaviour space. However, when starting from *tabula rasa* initial conditions, the controller is likely to produce actions which explore only a sub-space of the behaviour space. We have seen that even at the level of extremely simple robots we observe degenerated world models through a restricted controller. Active learning using virtual controller values is able to point out degeneracies in the world model and correct them by active exploration. Different strategies have been described in order to control the interplay of regular and active learning based behaviour. The two main ones are the MIDDLE strategy and the FARADAPT strategy. The first strategy causes a constant influence of active learning whereas the second strategy enables active learning for short phases only.

Conclusion and Outlook

This thesis describes a novel approach to machine learning and active learning called backward modelling and applies it to artificial feed-forward neural networks. The target application field is the sensorimotor loop of autonomous robots. We are especially interested in the cognitive bootstrapping problem arising from the concomitant learning of the controller and the world model starting from *tabula rasa* initialisation.

Backward modelling is a general concept, which is based on the inversion of the model represented by the learner. It generates virtual inputs by shifting the real inputs in order to minimise the output error. Using virtual inputs for learning induces fundamental effects such as spontaneous symmetry breaking and generation of responsive models, which have been elaborated in theory.

The application to feed-forward neural networks has been described in detail and different algorithms have been proposed. The two main ones are called BackModel and BackModelGiant. BackModel is a layer-wise approach, which treats the network as a sequence of smaller sub-networks. BackModelGiant is an over-all algorithm in the spirit of the standard backpropagation algorithm. In order to asses the algorithms and to verify the theoretical assertions experiments on different synthetic scenarios have been made. We are particularly interested in the learning behaviour with low weight initialisation, which concurs with the *tabula rasa* initialisation needed for unbiased cognitive bootstrapping. The generalised XOR problem called n-Parity has been used to show the effect of spontaneous symmetry breaking of backward modelling. The Encoder problem was used to verify the capability of the algorithms to deal with deep networks. Apart from that, a nonlinear regression problem called Square, was used to analyse the generalisation performance and the impact of noisy data. All experiments have shown the expected results and substantiate the theoretic predictions. Especially large benefits on highly symmetric environments and deep networks have been recorded.

Input shifts and virtual inputs can be used for active learning. Both pattern selection and pattern querying have been considered. For pattern selection we suggested a measure of information gain based on the size of the input shifts. For learning with a fixed dataset we could record minor improvements in learning speed on some scenarios. The success of active

learning depends on the structure of the task and our empirical results are not very satisfactory here. Backward modelling enabled a new way of pattern querying. Virtual inputs can be used to request patterns on the fly. Again we observed no improvements on the synthetic problems.

Finally, we have considered a simple robot with an adaptive world model. We observed degenerated models through a restricted controller. Backward modelling generates virtual controller values, which point out degeneracies in the world model and which can be used for direct active exploration. We have seen with simple example that these explorations correct irregularities of the world model quickly. The regularity of the world model is essential for the cognitive bootstrapping problem, because it enables the devising of new control strategies by the controller.

Different strategies have been described in order to control the interplay of controller based and active learning based behaviour. The two main ones are the MIDDLE strategy, which causes a constant influence of active learning, and the FARADAPT strategy, which enables active learning for short phases only.

We have seen that backward modelling can be used for classical learning schemes as well as for active learning scenarios in robot control. Particular benefits are obtained on deep networks and weight initialisation of arbitrary size. Further research can be directed to the application of backward modelling to recurrent networks. Since it is a general concept, the application to different numerical approaches such as Support Vector Machines seems possible as well. An application area for backward modelling might be the realm of cognitive tasks, where deep networks are usually used. The network inversion performed by backward modelling can be considered as a generalised form of receptive fields. Another promising direction is the investigation of more complex robots in order to obtain complex emergent behaviour.

Appendix A

Hanna

The neural network simulator used in this work is Hanna, the **H**askell **A**rtificial **N**eural **N**etwork simul**A**tor. It was developed by the author together with Patrick Scheibe and has grown to fairly large library with over 6 000 physical lines of code. The used programming language is Haskell (`http://www.haskell.org`) which is a pure functional language with static typing, higher-order functions, lazy evaluation and a lot more. It empowers fast prototyping and convenient programming of algorithms.

The simulator is about to appear at `http://www.hanna.de.ms` and eventually at `http://www.haskell.org/libraries`. It features flexible types of layers, neurons, connections and learning algorithms together with a powerful simulation handling. This includes classical learning schemas as well as active learning with offline or real world environments. Additionally, various ways for network analysis are provided, for example graphical output via the GraphViz tools (`http://www.graphviz.org`), a small real time GUI using wxHaskell, which is a binding for the wx-toolkit (`http://www.wxwindows.org`), and a Mathematica package for plots, which is extensively used in this work.

Appendix B

SimParEx

`SimParEx` stands for **Sim**ple **Par**allel **Ex**ecuter and is a tool to perform distributed task farming with arbitrary commandline programs. It was written by the author in Perl `http://www.perl.org` because the existing free programs do either require programs to be written in a specific language or require large administrative overhead. `SimParEx` is platform independent in the design and requires only an account with ssh key-authentication on each computer. The only constraint to the worker program is that it must be parameterised via commandline or input files.

`SimParEx` consists of a server and a client program. One computer in the network (TCP/IP) acts as the master and the rest are slaves. The client program is copied automatically to each slave with secure copy (`scp`) and started as a daemon with secure shell (`ssh`). After that the clients connect the server program at the master via a TCP-socket and request the configuration, the worker program, and one task after the other. The communication is done via the `HTTP` protocol. Results from the worker program are transmitted to the server the same way. The task definition is flexible and supports commandline construction and input file generation. Since the server program is a `HTTP`-server, the computation progress can be observed via web-interface.

The project is hosted by SourceForge (`http://sourceforge.net`) and can be found at `http://simparex.sourceforge.net` and `http://sourceforge.net/projects/simparex`.

Acknowledgements

I am grateful to my advisor Ralf Der for introducing me to the field of neural networks and active learning and for his constant help and support during the preparation of this work. Cooperation with him was highly motivating and I benefitted greatly from his careful advice.

It was a pleasure to work with the Robotic Group in Leipzig. Especially, I want to thank René Liebscher for the inspiring discussions and who generously shared knowledge on software related questions with me.

I would like to express my gratitude to my friend Patrick Scheibe, who developed parts of Hanna and the Mathematica package I used. His constant interest in the progress of my work and stimulating discussions have been very helpful.

Thanks to Allan Clark and Manuela Lindemeyer for carefully reading and commenting on the draft of this thesis.

My special thanks go to my family and my friends. The time of relaxation I could spend with them and the motivations I recieved have been essential during writing of this thesis.

Bibliography

[1] Janet Finlay and Alan Dix. *An Introduction to Artificial Intelligence*, chapter 4. UCL Press, Taylor & Francis Group, 1996.

[2] Herbert A. Simon. Why should machines learn? In *Machine Learning: An Artifical Intelligence Approach*, volume 1. Morgan Kaufmann, San Mateo, CA, 1983.

[3] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning: An Artificial Intelligence Approach*. Springer-Verlag, Berlin, 1983.

[4] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kaufmann, San Mateo, CA, 1986.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, March 1998.

[6] URL `http://en.wikipedia.org/wiki/Genetic\_algorithms`.

[7] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995. URL `citeseer.ist.psu.edu/cortes95supportvector.html`.

[8] A. Smola and B. Sch. A tutorial on support vector regression, 1998. URL `citeseer.ist.psu.edu/smola98tutorial.html`.

[9] M. Hasenjäger and H. Ritter. Active Learning in Neural Networks. URL `citeseer.ist.psu.edu/404108.html`.

[10] S. D. Whitehead. Complexity and cooperation in Q-learning. In L. A. Birnbaum and G. C. Collins, editors, *Proceedings of the Eighth International Workshop on Machine Learning*, pages 363–367. Morgan Kaufmann, San Mateo, CA, 1991.

[11] Mark Plutowski. *Selecting Training Exemplars for Neural Network Learning*. Ph.D. thesis, Department of Computer Science and Engineering, University of California, San Diego, CA, 1994.

[12] Mark Plutowski, Garrison Cottrell, and Halbert White. Experience with selecting exemplars from clean data. *Neural Networks*, 9:273–294, 1996.

[13] A. Röbel. Dynamic Pattern Selection: Effective Training Backpropagation Neural Networks. In *Proceedings of the International Conference on Artificial Neural Networks*, volume 1, pages 643–646. 1994.

[14] B. T. Zhang. Accelerated Learning by Active Example Selection. *International Journal of Neural Systems*, 5(1):67–75, 1994.

[15] A. Engelbrecht and A. Adejumo. A New Selective Learning Algorithm for Time Series Approximation using Feedforward Neural Networks. In VB Bajić and D Sha, editors, *Development and Practice of Articial Intelligence Techniques*, pages 29–31. 1999. URL `citeseer.ist.psu.edu/engelbrecht99new.html`. Proceedings of the International Conference on Articial Intelligence, Durban, South Africa.

[16] A. Engelbrecht and I. Cloete. Incremental Learning using Sensitivity Analysis. In *International Joint Conference on Neural Networks*. IEEE Press, Washington DC, USA, 1999. URL `citeseer.ist.psu.edu/engelbrecht99incremental.html`.

[17] A. P. Engelbrecht. Selective Learning using Sensitivity Analysis. In *Proceedings of the International Joint Conference on Neural Networks*, volume 11, pages 1150–1155. IEEE Press, Anchorage, Alaska, 1998.

[18] A. Adejumo and A. Engelbrecht. A comparative study of neural network active learning algorithms. In *Proceedings of the International Conference on Artificial Intelligence*, pages 32–35. 1999. URL `citeseer.ist.psu.edu/adejumo99comparative.html`.

[19] I. Guyon, N. Matić, and V. Vapnik. Discovering informative patterns and data cleaning. In U. M. Fayyad, editor, *Advances in Knowledge Discovery and Data Mining*, pages 181–220. AAI Press, Menlo Park CA, 1996.

[20] P.W.Munro. Repeat until bored: A pattern selection strategy. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 1001–1008. Morgan Kaufmann, San Mateo, CA, 1992.

[21] C. Cachin. Pedagogical pattern selection strategies. *Neural Networks*, 7:175–181, 1994.

[22] M. Hasenj and a Ritter. Active learning with local models, 1998. URL `citeseer.ist.psu.edu/hasenj98active.html`.

[23] J. N. Hwang, J. J. Choi, S. Oh, and R. J. Marks II. Query based learning applied to partially trained multilayer perceptrons. In *IEEE Transactions on Neural Networks*, volume 2, pages 131–136. 1991.

[24] C. Jensen, R. Reed, R. El-Sharkawi, J. Jung, R. Miyamoto, G. Anderson, and C. Eggen. Inversion of feedforward neural networks: Algorithms and applications, 1999. URL `citeseer.ist.psu.edu/jensen99inversion.html`.

[25] J. Kindermann and A. Linden. Inversion of multilayer nets. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 425–430. IEEE Press, New York, 1989.

[26] W. Kinzel and P.Ruján. Improving a network generalization ability by selecting examples. *Europhysics Letters*, 13:473–477, 1990.

[27] R. J. Williams. Inverting a connectionist network mapping by backpropagation of error. In *Proceedings of the Eighth Annual Conference of Congnitive Science Society*, pages 859–865. Lawrence Erlbaum, Hillsdale NJ, 1986.

[28] L. Atlas, D. Cohn, and R. Ladner. Training connectionist networks with queries and selective sampling. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 567–573. Morgan Kaufmann, San Mateo, CA, 1990.

[29] D. Cohn. Queries and exploration using optimal experiment design. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann, San Mateo, CA, 1994.

[30] G. Paass and J. Kindermann. Bayesian query construction for neural network models. In G. Tesauro, D. Touretzky, and T. K. Leen, editors, *Advances in Neural Processing Systems*, volume 7, pages 443–450. MIT Press, Cambridge, MA, 1995.

[31] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective Sampling Using the Query by Committee Algorithm. *Machine Learning*, 28(2-3):133–168, 1997. URL `citeseer.ist.psu.edu/freund97selective.html`.

[32] H. S. Seung, Manfred Opper, and Haim Sompolinsky. Query by Committee. In *Computational Learning Theory*, pages 287–294. 1992. URL `citeseer.ist.psu.edu/seung92query.html`.

[33] S. B. Thrun. The role of exploration in learning control. In David A. White and Donald A. Sofge, editors, *Handbook of intelligent control: nerual, fuzzy and adaptive approaches*. Van Nostrand Reinhold, Florence, Kentucky 41022, 1992.

[34] S. B. Thrun. Exploration in active learning, 1995. URL `citeseer.ist.psu.edu/thrun95exploration.html`.

[35] S. B. Thrun. Efficient Exploration in Reinforcement Learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1992. URL `citeseer.ist.psu.edu/article/thrun92efficient.html`.

[36] J. Schmidhuber. What's interesting? Technical Report IDSIA-35-97, IDSIA, 1997. URL `ftp://ftp.idsia.ch/pub/juergen/interest.ps.gz`. Extended abstract in Proc. Snowbird'98, Utah, 1998.

[37] J. Schmidhuber. Exploring the Predictable. In A. Ghosh and S. Tsuitsui, editors, *Advances in Evolutionary Computing*, pages 579–612. Springer, 2002.

[38] O. Kinouchi and N. Caticha. Optimal generalization in perceptrons. *Journal of Physics A*, 25:6243–6250, 1992.

[39] A. P. Engelbrecht. Selective Learning for Multilayer Feedforward Neural Networks. *Lecture Notes in Computer Science*, 2084:386–??, 2001. URL `citeseer.ist.psu.edu/499519.html`.

[40] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. MIT Press, Cambridge, MA, 1995.

[41] P Sollich. Query construction, entropy, and generalization in neural network models. *Physical Review E*, 49:4637–4651, 1994.

[42] D. MacKay. The Evidence Framework Applied to Classification Networks. *Neural Computation*, 4(5):720–736, 1992. URL `citeseer.ist.psu.edu/mackay92evidence.html`.

[43] D. MacKay. Information-Based Objective Functions for Active Data Selection. *Neural Computation*, 4(4):590–604, 1992. URL `citeseer.ist.psu.edu/47461.html`.

[44] D. Cohn. Neural network exploration using optimal experiment design. In *Neural Networks*, volume 9, pages 1071–1083. 1996.

[45] L. M. Belue, K. W. Bauer, and D. W. Ruck. Selecting optimal experience for multiple output multilayer perceptrons. *Neural Computation*, 9:161–183, 1997.

[46] B. Eisenberg and R. L. Rivest. On The Sample Complexity Of Pac-Learning Using Random And Chosen Examples. In M. Funk and J. Case, editors, *Processings of the Third Annual Workshop on Computational Learning Theory*, pages 154–162. University of Rochester NY, Morgan Kaufmann, San Mateo, CA, August 1990.

[47] Sanjeev R. Kulkarni, Sanjoy K. Mitter, and John N. Tsitsiklis. Active Learning Using Arbitrary Binary Valued Queries. *Machine Learning*, 11:23–35, 1993.

[48] E. B. Baum and K. Lang. Query learning can work poorly when human oracle is used. In *International Joint Conference on Neural Networks*. Beijing, China, 1992.

[49] A. Engelbrecht and R. Brits. A clustering approach to incremental learning for feedforward neural networks. In *Proceedings of the International Joint Conference Neural Networks*, volume 3. 2001. URL `citeseer.ist.psu.edu/485864.html`.

[50] Y. Baram, R. El-Yaniv, and K. Luz. Online choice of active learning algorithms, 2003. URL `citeseer.ist.psu.edu/luz03online.html`.

[51] Randall C. O'Reilly and Yuko Munakata. *Computational Explorations in Cognitive Neurosience, Understanding of the Mind by Simulating the Brain*. MIT Press, Cambridge Massachusetts, 2000.

[52] P. J. Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral science*. Ph.D. thesis, Harvard University, Cambridge, MA, 1974.

[53] Andreas Zell. *Simulation Neuronaler Netze*. Addison-Wesley, 1994.

[54] D. E. Rummelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.

[55] Martin Riedmiller and Heinrich Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP algorithm. In *Proc. of the IEEE Intl. Conf. on Neural Networks*, pages 586–591. San Francisco, CA, 1993. URL `citeseer.ist.psu.edu/riedmiller93direct.html`.

[56] Mark Jurik. Backpercolation, 1991. Paper distributed by Jurik Research and Consulting, PO 2379, Aptos, CA 95001, USA.

[57] S. E. Fahlman. An empirical study of learning speed in back-prop networks. In D. Touretzky, G.H inton, and T. Sejnowski, editors, *Connectionists Models Summer School*. Carnegie Mellon University, Morgan Kaufmann, 1988.

[58] P. J. Werbos. Backpropagation: Past and future. In *Processing of the International Conference on Neural Networks*, volume I, pages 343–353. IEEE Press, New York, July 1988.

[59] R. A. Jacobs. Increased Rates of Convergence Through Learning Rate Adaptation. *Neural Networks*, 1:295–307, 1988.

[60] W. B. Cannon. *The wisdom of the body*. Norton, New York, 1939.

[61] R. Der, U. Steinmetz, and F. Pasemann. Homeokinesis - A new principle to back up evolution with learning, 1999. URL `citeseer.ist.psu.edu/der99homeokinesis.html`.

[62] Ralf Der and Thomas Pantzer. Emergent Robot Behavior From the Principle of Homeokinesis. URL `citeseer.ist.psu.edu/231153.html`.

[63] R. O'Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm, 1996. URL `citeseer.ist.psu.edu/35164.html`.

[64] R. Der. Self-Organized Acquisition of Situated Behavior. *Theory Bioscience*, 120:179–187, 2001.

[65] Arthur Flexer. Statistical Evaluation of Neural Network Experiments: Minimum Requirements and Current Practice. Technical Report OEFAI-TR-95-16, The Austrian Reseach Institute of Artificial Intelligence, Schottengasse 3, A-1010 Vienna, Austria, 1995. URL `citeseer.ist.psu.edu/flexer94statistical.html`.

[66] Lutz Prechelt. Some Notes on Neural Learning Algorithm Benchmarking. *Neurocomputing*, 1995.

[67] Lutz Prechelt. Early Stopping-But When? In *Neural Networks: Tricks of the Trade*, pages 55–69. 1996. URL `citeseer.ist.psu.edu/article/prechelt97early.html`.

[68] Erik Hjelmas and Paul W. Munro. A Comment on the Parity Problem. URL `citeseer.ist.psu.edu/hjelmas99comment.html`.

[69] Hohil, Liu, and Smith. Solving the N-bit Parity Problem Using Neural Networks. *NNETS: Neural Networks*, 12, 1999. URL `citeseer.ist.psu.edu/hohil99solving.html`.

[70] A. D. Sontag. Feedforward nets for interpolation and classification. *Journal of Computer and Systems Science*, 45:20–48, 1992.

Eigenständigkeitserklärung

Ich versichere, daß ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, 15.02.2005