Graz University of Technology

Slowness as a feature extraction principle for Reinforcement Learning

Master's Thesis

in Computer Science

by

Tobias Hönisch

University:	Graz University of Technology
Institute:	Institute for Software Technology
Supervising:	Prof. Minorua Asada
	Prof. Franz Wotawa
Handed in:	December 2010

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)

Acknowledgments

It is a pleasure to thank those who made this thesis possible. I owe my deepest gratitude to Prof. Asada for his guidance and supervision of this thesis. I want to thank Prof. Wotawa and Dr. Steinbauer who provided me with this great opportunity to write this thesis at the Osaka University. I also want to show my gratitude to all members of Erato and the Asada Laboratory who welcomed me so heartily and integrated me into their team.

Especially I want to thank Joschka Boedecker for his help and motivation in writing this thesis, without you this thesis would have not been possible.

Lastly I offer my regards to all of those who supported me in any respect during the writing of this thesis.

Tobias Hönisch

Abstract

Slow Feature Analysis is an unsupervised feature extraction algorithm that extracts non-linear features based on slowness. In many cases slowness is a good indicator for the usefulness of information and can therefore be used to extract features out of huge input spaces. We want to show that these extracted slow features are useful for Reinforcement Learning. Slow Feature Analysis does not have to be fine-tuned or adapted to a problem and it always finds the optimal solution in a certain function space within a single iteration. Therefore it does not have convergence issues. With these properties it is very suitable for an autonomous preprocessing step before applying Reinforcement Learning on a control task. The complete learning system can be applied to complex control tasks and fares comparably well when compared to Reinforcement Learning on the underlying driving force, a high level representation of the important information contained in the huge input space.

In this thesis we will show the performance of the proposed learning system on two different control tasks. For one task we will show that slowness is not necessarily the best indicator for certain features to be extracted. For the second task we will show that the learning system is able to outperform the learning system consisting of the same Reinforcement Learning algorithm being applied to a high level representation of the contained information.

keywords: slow feature analysis, reinforcement learning, unsupervised feature extraction

Contents

Chapter 1

Introduction

The effort to use a machine learning algorithm depends largely on the time needed to fine-tune its parameters. Knowledge of the field of application and expertise of the algorithm is required to do that. Until now different research directions like Computer Vision or Signal Processing have developed many sophisticated methods to extract useful features with different goals in mind. Often these so-called feature extraction methods are suitable for specialized tasks only and choosing one in a suitable way requires knowledge of the task at hand and does not work in an unsupervised manner.

Autonomous learning has always been of major interest in the field of Artificial Intelligence. Still today many machine learning solutions require to be tuned and adapted by an expert to fit the special requirements of a certain problem setup. For example the hole field of Artificial Neural Networks, with adaptation possibilities not only in terms of network type, but also in layer depth, connectivity or its activation function. While much research is been done on finding more efficient algorithms, the need to handle bigger and bigger data sets calls for unsupervised feature extraction algorithms. Slow Feature Analysis(SFA) (see [WS02]) is an unsupervised feature extraction method based on slowness. The slowness principle comprises the idea that important information spread over many input features, changes slowly in comparison to the change of information encoded in a single feature. This slowness principle might well be used in our brain as well. It has been shown that the usage of SFA can lead to the creation of place, head-direction, and spatial-view cells[FSW07]. These cells decode basic information out of a big video signal which is thought to be needed for self localization, orientation and navigation. Slowness is an important aspect in the self organization in the visual cortex[WB03]. This makes the algorithm a plausible explanation for the preprocessing of the huge data the brain is confronted with in a constant matter.

SFA has been used in different architectures to show different aspects of its usability. We want to show that the combination of SFA and Reinforcement Learning (RL) can be used successfully for more complicated control tasks. SFA has been analyzed and used in different papers. Please see [LWW10] for the use of a similar system on the Morris water maze task. The use of SFA with receptive fields can be used to avoid the curse of dimensionality

[FSW07]. Another possibility to avoid this problem is the use of the kernel trick (see [BM02]). Please see [SW08] for a detailed mathematical analysis of SFA. Another work has been done by [AS09] for the usage of an artificial neural network with a reservoir with the SFA. For details of these related works please see chapter Related Work 5. Many of these works provide interesting thoughts for the extension of the learning system in this thesis and are discussed in the Future Work section (see 6) of this thesis for its applicability in this context. We therefore propose SFA as preprocessing stage for a later RL in a control task.

Reinforcement Learning being the second part of the proposed learning system is a well researched area of machine learning. See [SB99] for a good introduction and [KLM96] for a survey on the topic. The idea of RL is the idea to give a learning agent feedback in a loose manner. It is not told how to achieve or perform, but given a reward telling it indirectly about its performance. Many algorithms have been proposed in this field of research. Direct Policy Search algorithms treat the RL problem as an optimization problem, in which the nature of the environment and the value function is not known and is not constructed during the learning. Its goal is to optimize the maximum reward in a black-box manner.

The usage of this type of RL algorithm was chosen for its performance on the used control tasks. Many other RL algorithms are usable and should be considered for future work.

1.1 Motivation

The information we can access and use is growing every day in an exponential way. Our own brain is very good in extracting useful features out of huge input spaces, which are provided by our senses. So far we have not been able to copy or even understand the inner working of our brain in a sophisticated way. Many theories and algorithms have been and still are proposed, which are thought to be plausible as being applied in our brain. So far most learning systems need to be fine-tuned and adapted closely to the desired task at hand. An expert is required who chooses 'good' features to be extracted for a later learning stage. The slowness principle[WS02] is one idea which is thought to be used in our brain[WB03]. With SFA we have an unsupervised feature extraction algorithm, which relies solely on slowness to extract relevant features.

So it's only the next logical step to ask how to use the extracted features. RL is a well established field in machine learning and provides us with many well understood algorithms to solve control tasks. It is used as a learning process in the brain of animals, which are in the need to model future reward to be able to manipulate its causal environment in a predicable way[ARL$^+$09], and can be applied to many practical control problems. By using RL with SFA we have a potentially completely autonomous learning system, which is capable of learning control tasks. For the system to function properly we only need to define a reward function and let it learn in a trial and error way on the provided slow features.

1.2 Problem Definition

The problem setup is a control task which is meant to test the proposed learning system. The use of a big input space $\mathbf{x}(t)$ together with RL will require a suitable feature extraction method. In this case the input space will consist solely of a video signal. All necessary information needs to be extracted out of this video signal to perform RL.

Two different control tasks for the problem setup were chosen. One being the classic Cart-pole experiment. The other is the game Pong. Please see chapter Implementation 3.4 for details of the implementation of the two control tasks. Both control tasks have a single action dimension $a \in A$. For the Cart-pole experiment it is the exerted force to move the cart left and right and for the Pong experiment it is the movement direction of the paddle. The goal of the control task is encoded in a reward function R_t. We can not use RL directly on this control task, because of the huge dimension space of $\mathbf{x}(t)$. To solve this problem we need to extract meaningful information out of $\mathbf{x}(t)$. There are many ways for supervised and unsupervised feature extraction. Examples for unsupervised feature extraction are Principal Component Analysis (PCA) or Independent Component Analysis (ICA). For supervised feature extraction one can think of the use of an Artificial Neural Network (ANN) trained as an edge detector. For this thesis we chose SFA as an unsupervised feature extraction algorithm for the preprocessing.

The implemented learn system consists of an environment, a SFA preprocessing stage, and the agent. The environment provides the high dimensional image stream. The SFA preprocessing stage consists of five layers. The first layer performs a linear SFA directly on the input signal $\mathbf{x}(t)$. Each of the other four layers performs polynomial expansion on the input space before performing SFA again. With every added layer we polynomially expand the search space in which the slow features can be found. By performing the polynomial expansion four times, we increase the search space from linear functions P_1 to the polynomial space P_{16}. This polynomial expansion is needed to find features which are more complex than linear combinations of the input signal. We only use the slowest 32 features on each layer.

When the SFA is done, we have 32 slow features. We used the slowness indicator $\eta(\mathbf{x})$ to determine the number of useful features for the RL (see the Slowness Analysis section of the corresponding experiments in the Results chapter 4). Only the slowest 8 features were used for the RL agent. The agent used a Black-box Optimizer approach to solve the control task. Fitness Expectation Maximization (FEM) was used at this stage. Fitness Expectation Maximization is explained in detail in 2.1.3.3.

1.3 Goal

The goal of the thesis is to analyze the proposed learning system in depth. We will provide the system with a huge input signal and evaluate its performance with the two proposed control tasks (see chapter Implementation 3). We want

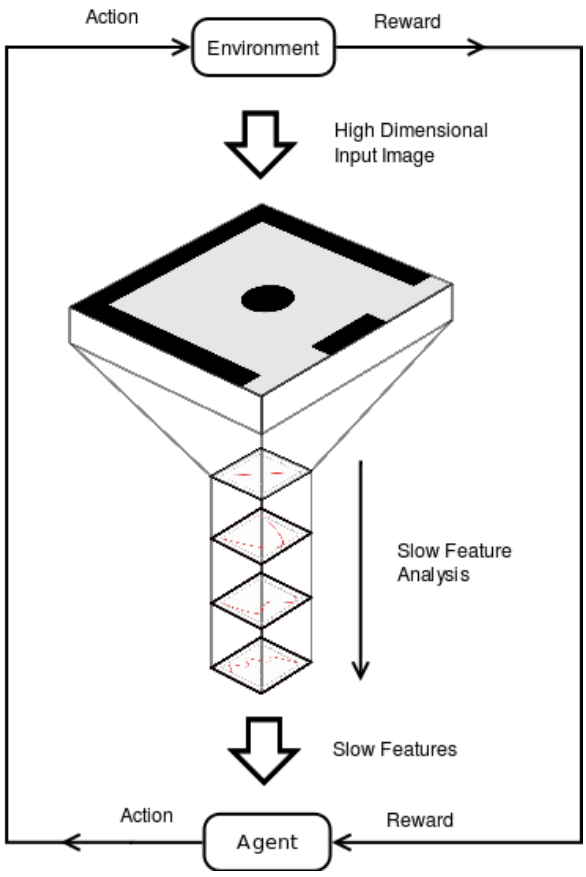


Figure 1.1: Learning System - for details, see main text

to determine its overall applicability and usability. Therefore we will analyze the impact of different parameter combinations on the overall performance. Finally we want to compare it to other unsupervised feature extraction methods and show its advantages, but also its limitations and how to get around them.

1.4 Structure of the Thesis

This thesis consists of seven parts, which will be presented in a logical order:

- **Introduction:**
  The introduction contains an overview of the thesis and the motivation to write it. The learning system and the problem setup will also be explained.

- **Theoretical Foundation:**
  This section will explain all the necessary information, which is needed to understand the idea of the thesis. First the idea of the Slow Feature Analysis from [WS02] is introduced and its idea of unsupervised feature

extraction, based on slowness is explained.

The second part needed for this thesis is Reinforcement Learning, which is one of the major fields of machine learning in the context of robotics. It's simple idea is presented and algorithms in this field which were used for the thesis are explained and detailed.

- **Implementation:** The details of the implementation will be presented in this section. Two approaches were tried. The use of a RoboCup Soccer simulator was the first try and will only be covered in short. We will give reasons why it was of no use and why we stopped using it. An own simulator was implemented later. This second approach, being the one which was used to show the usefulness of the thesis idea, will be explained and detailed. Both experiments, the Cart-pole experiment and the Pong experiment are detailed in here too. A short overview of the used libraries will also be given in here.

- **Results and Discussion:**
  For the results we will present reasons for the success and also failure of certain aspects in the two experiments which were used. Its limitations will also be presented.

- **Related Work:**
  In the Related Work section we will focus on the comparison to other unsupervised feature extraction methods which were used in conjunction with RL.

- **Future Work:**
  This paper is meant to show the usefulness of the proposed learning system. In this section we want to give advice on further research which will be conducted. It also proposes some ideas on optimizing the given system and where future research can lead us.

- **Conclusion:**
  The final thoughts on this thesis, wrapped up in a short paragraph.

Chapter 2

Theoretical Foundation

Before getting into the details of our system we want to describe the algorithms on which the system is build.

2.1 Reinforcement Learning

Reinforcement Learning is the idea to teach a system without giving it the correct answer, but giving it a reward telling it indirectly what is wrong and right. It therefore lies in between Supervised Learning, in which case the correct answer is known and Unsupervised Learning, in which case no feedback is given to the agent.

RL algorithms are applied in many places in nature which perform learning. Agents, in this case kids, pets or normal people will seldom get concrete answers what to do, but rather a reward for a certain behavior. Even classic supervised learning, which kids will encounter in school, comprises a lot of RL. In this case consider the grading as the reward signal which contains the information on how to please the teacher, which might not only include learning the necessary information by heart, but also the presentation or necessity of certain information. Actually one could say that social interaction and human communication comprises a lot of RL as we are constantly being judged and evaluated for our interactions and then fed back a very indirect feedback with body language and other communication means. The human or agent therefore has to figure out which action or action combination led to this certain feedback or so-called reward.

This chapter will closely follow [SB99]. First the basic idea will be presented and later the details of using Direct Policy Search in RL.

2.1.1 Basic Idea

A RL system consists of an agent and an environment(see figure 2.1). The agent interacts with the environment in discrete time-steps $t \in \{0, 1, 2, \dots\}$. For every time-step t the agent will be in state $s_t \in S$ and takes an action a_t. It will observe a reward r_{t+1} and find itself in the next state s_{t+1}. The goal is

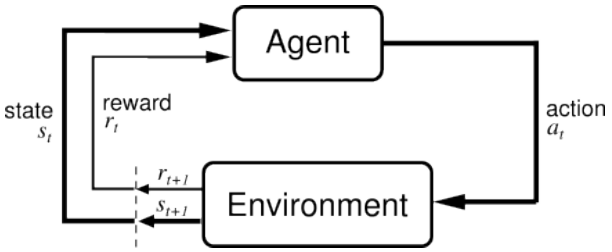


Figure 2.1: RLSystem

it to maximize the discounted future reward R_t.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

The discount factor $0 \leq \gamma \leq 1$ discounts the future to make the current reward gained more important than future reward. A $\gamma = 1$ can be used in episodic tasks, where discounting is not needed, and will give equal importance for the gained reward to all encountered states. A $\gamma < 1$ will prioritize actions in the short past over actions long ago. The RL Problem therefore is to find a policy π with $a_t = \pi(s_t)$ which maximizes R_t. We will call this policy π^*. There are many different concepts on how to find this policy π^*.

2.1.2 Value Function Approximation

These kind of approaches try to approximate the value function V^π. This value function V^π is a function which maps a certain state s to the expected total future reward E_π for being in that state s and following policy π.

$$\begin{aligned} V^\pi(s) &= E_\pi \left\{ R_t | s_t = s \right\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \end{aligned} \tag{2.2}$$

$V^\pi(s)$ describes the discounted total future reward R_t under the condition of being in state s_t and following policy π. Although the Value Function $V^\pi(s)$ is enough to define optimality for policy π^*, we want to define the action value function $Q^\pi(s, a)$ to make further formulas easier to read. As its parameters are a and s, we can directly deduce a the greedy policy π from $Q^\pi(s, a)$. With only $V^\pi(s)$ we would need to look one step ahead to do the same.

$$\begin{aligned} Q^\pi(s, a) &= E_\pi \left\{ R_t | s_t = s, a_t = a \right\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_1 = a \right\} \end{aligned} \tag{2.3}$$

Therefore $Q^\pi(s, a)$ defines the value for the discounted total future reward R_t under the condition of being in state s_t, choosing action a and following policy π for all further encountered states. Therefore we can derive the optimal policy π^* as:

$$\begin{aligned} \pi^* &= \arg\max_a Q^\pi(s, a) \\ &= \arg\max_a E \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a \right\} \end{aligned} \tag{2.4}$$

For those who are interested in how the algorithms work to find policy π^* please read [SB99].

2.1.3 Direct Policy Search

For this thesis two algorithms were used. Both belong to the group of Direct Policy Search algorithms. These kind of algorithms try to find a policy π in a black-box manner without trying to approximate the Value Function $V^{\pi}(s)$. Algorithms of this category are in fact Stochastic Optimization Algorithms. Therefore any optimization algorithm can be used for Direct Policy Search.

These algorithms perform local search. Optimality can therefore not be guaranteed. They perform reasonable well on the chosen dataset and were chosen as reference algorithms. Two different groups can be differentiated. Gradient-based and gradient-free algorithms. We only have data points for the estimation of an underlying policy function. Therefore the gradient can not be deduced directly and gradient-based algorithms have to rely on noisy estimates of the gradient. The other group, gradient-free algorithms don't rely on the gradient at all. Evolutionary Algorithms (see survey in [MSG99], Simulated Annealing (see Adaptive Simulated Annealing in [API]) and Cross Entropy Search[MRG03] algorithms, just to name a few, belong to this group of optimization algorithms.

2.1.3.1 Black-box Optimization in RL

The objective in Black-box Optimization is to maximize an unknown fitness function $f : \mathbb{R}^n \to \mathbb{R}$. The function f is assumed to be unknown or undefined and its evaluation to be possible, but costly. We therefore want to find a solution candidate with high fitness by using a limited number of function evaluations. Exhaustive search in the fitness landscape is considered infeasible.

For RL we define the fitness function to be the total reward R_{π} gained in one episode while following policy π:

$$f(\pi) = R_{\pi} = \sum_{k=0}^{N-1} r_{k+1} \tag{2.5}$$

This definition of f converts the RL problem into a supervised learning problem, where the parameters of π are adapted after each episode.

2.1.3.2 Stochastic Hill Climber

Hill Climbing is one of the simplest kind of Optimization Algorithms. It will start at a random position and move uphill until it reaches a local minimum. Stochastic Hill Climber (see algorithm 1) is an optimized version of this Hill Climber algorithm. It will stochastically accept a worse solution given a temperature parameter.

2.1.3.3 Fitness Expectation Maximization

Fitness Expectation Maximization is an Expectation Maximization algorithm. This explanation follows closely the description in [WSPS08]. It is a heuristic approach which tries to maximize the expected fitness $J = \mathbf{E}_{\mathbf{z}}[f(\mathbf{z})]$ with $\mathbf{z}$

Algorithm 1 Stochastic Hill Climber

1: $bestSolution = random$
2: **for** $round = 1$ to $maxrounds$ **do**
3: $\quad challenger = bestSolution.mutate()$
4: $\quad$ **if** $evaluate(challenger) > evaluate(bestSolution)$ **then**
5: $\quad\quad bestSolution = challenger$
6: $\quad$ **else if** $e^{\left|\frac{eval(challenger)-eval(bestSolution)}{temperature}\right|} < random$ **then**
7: $\quad\quad bestSolution = challenger$
8: $\quad$ **end if**
9: **end for**

being a certain search point in the fitness landscape. Search points $\mathbf{z}$ are found using a search policy π:

$$\pi(\mathbf{z}|\theta) = \mathcal{N}(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}) = ((2\pi)^{n/2} |\mathbf{\Sigma}|^{1/2})^{-1} \exp\left[-\frac{1}{2}(\mathbf{z}-\mathbf{x})^{\mathbf{T}}\mathbf{\Sigma}^{-\mathbf{1}}(\mathbf{z}-\mathbf{x})\right]$$

This equals the probability density for search point $\mathbf{z}$ given policy π, which is a Gaussian with mean $\mathbf{x}$ and covariance matrix $\mathbf{\Sigma}$. The defining parameters of policy π are combined in the vector θ.

$$\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$$

By simply trying to optimize the objective function J we will find that the algorithm is either too aggressive with little knowledge or converges prematurely. An utility function $u(\mathbf{x})$ is introduced to combat this problem. Its requirement is that it scales monotonically with f, is semi-positive $u(f) \geq 0$ and integrates to a constant. We will get a new objective function:

$$J_u(\theta) = \int p(\mathbf{z}|\theta) u(f(\mathbf{z})) d\mathbf{z}$$

In this context a rank based utility function is chosen, as it fares well with a broad spectrum of problems:

$$u_k = u(f(\mathbf{z}_k)|f(\mathbf{z}_{k-1}), \ldots, f(\mathbf{z}_{k-N}))$$

It is piecewise linear. After ranking all values $\mathbf{z}$ by their corresponding fitness values, 0 is assigned to the worst $N - m$ samples and a linear distribution from 0 to 1 to the top m samples.

Adding it together we get the Fitness Expectation Maximization algorithm. See algorithm 2 (taken from [WSPS08])

2.2 Slow Feature Analysis

Slow Feature Analysis is a fairly new idea introduced by [Wis98] for unsupervised extraction of *meaningful* information from temporal data. The main idea

Algorithm 2 Fitness Expectation Maximization

use shaping function u, batch size N, forget factor α
$k \leftarrow 1$
initialize search parameters $\theta^{(k)} = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$
repeat
    draw sample $\mathbf{z}_k \sim \pi(\mathbf{x}, \mathbf{\Sigma})$
    evaluate fitness $f(\mathbf{z}_k)$
    compute rank-based fitness shaping $u_k = u(f(\mathbf{z}_k)|f(\mathbf{z}_{k-1}, \ldots, f(\mathbf{z}_{k-N}))$
    $\mathbf{x} \leftarrow (1 - \alpha u_k)\mathbf{x} + \alpha u_k \mathbf{x}$
    $\mathbf{\Sigma} \leftarrow (1 - \alpha u_k)\mathbf{\Sigma} + \alpha u_k (\mathbf{x} - \mathbf{z_k})(\mathbf{x} - \mathbf{z_k})^{\mathbf{T}}$
    $k \leftarrow k + 1$
until stopping criterion is met

is that the world changes slowly compared to the fast changes of the sensor readings trying to capture that information. Every sensor has only a very limited local view. For example one pixel in a camera or a light receptor in an human eye. By combining many of these sensors, useful information can be extracted. It has been shown that SFA can be used for the self organized formation of place cells, head-direction cells, and spatial-view cells as they are present in the brain of rats [FSW07]. Therefore, SFA is an Unsupervised Feature Extraction Algorithm that can be used to extract useful information for further learning stages in a cognitive process.

2.2.1 Slowness Principle

Today we have access to an huge amount of data.

> "There was 5 exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days, and the pace is increasing..." (Eric Schmidt, Google CEO, 04.08.2010)

Most of this data is raw and its information content needs to be processed to be useful in the machine learning context. The computational view of perception is the extraction of environmental information out of quickly varying sensor data. This environmental information is spread over many of these sensors, which encode very localized information.

For example the relevant information in a video stream of zebras would be the object identity and the object position. The signal of one of the sensors would vary quickly as the stripes of the zebras move through its field of view. The object identity and the object position on the other hand vary much slower. An algorithm can be formulated to extract these slow varying features out of a fast varying input space.

2.2.2 Algorithm

This chapter will closely follow [WS02] to describe the algorithm and its requirements. The mathematical optimization problem is defined first. An algorithm

is constructed for the simpler case of considering only linear combinations of the input signal. The limitation of finding only linear combined functions can be avoided by expanding the input signal nonlinearly in a preprocessing stage.

2.2.2.1 Problem Definition

Given an input signal: $x(t) = [x_1(t), x_2(t), \ldots, x_I(t)]$ with $t \in [t_0, t_1]$. We want an output function: $g(x) = [g_1(x), g_2(x), \ldots, g_J(x)]$ which generates $y(t) = [y_1(t), y_2(t), \ldots, y_J(t)]$ with $y_j(t) := g_j(x(t))$ from $x(t)$ instantaneous to achieve slowness:

$$\Delta_j := \Delta(y_j) := \left\langle \dot{y}_j^2 \right\rangle \rightarrow \min$$

The following constraints are imposed:

$$\langle y_j \rangle = 0 \text{ (zero mean)} \tag{2.6}$$

$$\left\langle y_j^2 \right\rangle = 1 \text{ (unit variance)} \tag{2.7}$$

$$\forall j' < j : \left\langle y_{j'} y_j \right\rangle = 0 \text{ (decorrelation)} \tag{2.8}$$

Angle brackets indicate temporal averaging $\langle f \rangle := \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} f(t)dt$. Unit variance 2.7 and decorrelation 2.8 are imposed to prevent trivial solutions. The unit variance constraint prevents the construction of constant signals and the decorrelation constraint is needed to enforce distinctness of the found slow features. The zero mean constraint 2.6 is for convenience only. It should be noted that a instantaneous response is required for the slow features y_j. Every response in time $y(x_t)$ depends only on the current value x_t and therefore prohibits the creation of a low pass filter.

This optimization problem is one of variational calculus and is in general difficult to solve. However, if we consider only linear combinations in the optimization step, the problem simplifies significantly.

2.2.2.2 Algorithm

A straight forward algorithm can be constructed, by considering only linear combinations. A preprocessing stage is proposed to nonlinearly expand the input signal before applying the algorithm to find higher order slow features. We apply $h_j(x)$ to $x(t)$ for the non-linear expansion. This will give us the nonlinearly expanded signal $z_j(t) = h_j(x(t))$. The optimization will simplify insofar that we only need to optimize the weights w_j in $y_j(t) = g_j(x(t)) = w_j^T h(x(t)) = w_j^T z(t)$. This will give us:

$$\Delta(y_j) := \left\langle \dot{y}_j^2 \right\rangle = w_j^T \left\langle \dot{z} \dot{z}^T \right\rangle w_j \rightarrow \min \tag{2.9}$$

We further choose $z(t)$ with 0 mean and a unit covariance matrix: :

$$\langle y_j \rangle = w_j^T \underbrace{\langle z \rangle}_{=0} = 0$$

$$\left\langle y_j^2 \right\rangle = w_j^T \underbrace{\left\langle z z^T \right\rangle}_{=I} w_j = w_j^T w_j = 1$$

$$\forall j' < j : \left\langle y_{j'} y_j \right\rangle = w_{j'}^T \underbrace{\left\langle z z^T \right\rangle}_{=I} w_j = w_{j'}^T w_j = 0$$

The imposed constraints will be already fulfilled if we choose the weight vectors to be an orthonormal set of vectors. Therefore the solution is the set of normed vectors of the matrix $\left\langle \dot{z} \dot{z}^T \right\rangle$ which correspond to the smallest eigenvalues. The vectors are sorted ascendingly by their corresponding eigenvalues. The one with the smallest eigenvalue will be the slowest extracted feature.

2.2.2.3 Measuring Slowness

We want to use a more intuitive measure for judging slowness than using the Δ values directly. We will use slowness measure proposed in
[WS02] for comparing the slowness of features with each other:

$$\eta(x) = \frac{T}{2\pi} \sqrt{\Delta(x)} \tag{2.10}$$

This measure is a more intuitive measure describing how many oscillations a pure sine wave with the same $\Delta(x)$ would have. A smaller $\eta(x)$ indicates a slower signal.

Algorithm 3 Slow Feature Analysis

1. **Input signal:**

$$\tilde{x}(t)$$

2. **Normalization:**

$$\begin{aligned} x(t): &= [x_1(t), \ldots, x_I(t)]^T \\ \text{with } x_i(t): &= \frac{\tilde{x}_i(t) - \langle \tilde{x}_i \rangle}{\sqrt{\langle (\tilde{x}_i - \langle \tilde{x}_i \rangle)^2 \rangle}}, \\ \text{so that } \langle x_i \rangle &= 0, \\ \text{and } \langle x_i^2 \rangle &= 1. \end{aligned}$$

3. **Nonlinear Expansion:**

$$\tilde{z}(t) := \tilde{h}(x(t))$$

4. **Sphering**:

$$\begin{aligned} z(t) &:= S(\tilde{z}(t) - \langle \tilde{z} \rangle), \\ \text{with } \langle z \rangle &= 0 \\ \text{and } \langle zz \rangle &= I. \end{aligned}$$

S is the shearing matrix and is determined by the use of Principal Component Analysis on the matrix $(\tilde{z}(t) - \langle \tilde{z} \rangle)$.

5. **Principal Component Analysis:**

$$\begin{aligned} w_j : \langle \dot{z}\dot{z}^T \rangle w_j &= \lambda_j w_j \\ \text{with} \quad \lambda_1 \leq \lambda_2 \leq &\ \ldots \ \leq \lambda_J, \end{aligned}$$

we get:

$$\begin{aligned} g(x) &:= [g_1(x), \ldots, g_J(x)]^T \\ \text{with} \quad g_j(x) &:= w_j^T h(x))) \end{aligned}$$

and output signal:

$$\begin{aligned} y(t) &:= g(x(t))) \\ \text{with} \quad \langle y \rangle &= 0, \\ \langle yy^T \rangle &= I, \\ \text{and} \quad \Delta(y_j) &= \lambda_j. \end{aligned}$$

λ_j are the eigenvalues.

Chapter 3

Implementation

3.1 Simulation Environment

Two different approaches were implemented to test the system. The first approach was the use of the Robocup Soccer Simulator. It turned out that the speed of the simulator was not sufficient. It had to render a 3D scene for every frame and could not provide the speed, needed to perform millions of evaluations. A simple graphic simulator, directly producing 2D images, was implemented and used instead to speed up the simulation.

3.1.1 Robocup Soccer Simulator

The Robocup is an international competition in the field of robotics with the goal to build a fully autonomous humanoid robotic soccer team that can beat the human world champion team in 2050. One of the Robocup leagues is the 3D soccer simulation league with the Simspark simulator [OORR04, BA08] providing the environment for virtual soccer matches. It's goal is it to tackle software challenges, which arise when building real physics robots. The Simspark soccer simulator is a separate process which communicates with the agents via TCP. This simulator can easily be adapted and used for other simulation problems. For this thesis we implemented a cart pole agent to be used in the simulation, with the goal of using the proposed learning system to teach it to balance the pole. The fact that the simulation was done in 3D was the limiting factor. Therefore a simpler 2D simulator was implemented called Simple Graphics Simulator.

3.1.2 Simple Graphics Simulator

As the Robocup Soccer Simulator did not provide enough speed, the Simple Graphics Simulator was implemented. It uses the underlying driving force to create the images. The Python Imaging Library was used for the image generation. It is a 2D image simulator which can produce the needed video stream with a much higher speed than the RoboCup Simulator. It provides the video stream of images in high speed, is adaptable in image size and can easily be extended for different control tasks.

3.2 Reinforcement Learning using Slow Features

In this thesis we propose to use SFA with RL. The usefulness of this approach has also been demonstrated in [LWW10]. The system consists of two training phases. The first phase will be a SFA training phase. There is no RL during this first phase. The second phase is the RL phase. The RL will then try to find an optimal policy by using the slow features found by the SFA.

First Phase - SFA

During this phase we generate data by following a fixed policy π_{sfa}. The policy π_{sfa} was chosen to be a random walk. Its requirement is to include most possible state transitions. Otherwise the RL phase will not be able to find an optimal policy π^*. The SFA on the generated data is performed in multiple layers. The first layer consists of adding noise to the data signal and then performing SFA to extract linear features. The other four layers consist of adding noise, polynomially expanding the input space and then performing SFA. Every one of these layers will increase the function space for the slow features to be found polynomially. The noise is added to make the calculations numerically stable and also to generalize. By using the SFA four times, we are able to find slow features in the polynomial space P_{16}. We only consider the slowest 32 features found by the SFA on each layer. For the last layer we use the slowness measure $\eta(x)$ (see the SFA section 2.2 in the Theoretical Foundation chapter for details) to find the appropriate number features which were faster than the input signal.

Second Phase - Reinforcement Learning

The second phase is the RL phase. We use the found slow features and the reward provided by the environment to train the agent. The agent uses a direct policy search approach to optimize the weights in a linear two layer artificial neural network. The second layer consists of one neuron, as only one continuous action dimension exists. The artificial neural network is therefore a linear combination of the provided slow features.

For the direct policy search we use the Fitness Expectation Maximization [WSPS08] algorithm. It tries to maximize the expected fitness of a defined fitness function $J = \mathbf{E}_{\mathbf{z}}[f(\mathbf{z})]$. The parameter $\mathbf{z}$ is the vector consisting of the weights for the ANN. The fitness function is defined as:

$$J(\pi_s) = R_{\pi_s} = \sum_{t=0}^{\infty} r_t | \pi = \pi_s \tag{3.1}$$

pi_s is the current search policy. R_{π_s} is therefore the total reward gained in one episode. R_{π_s} is a rational number, as there is a limit on the maximum number of time steps t per episode.

Please see figure 3.1 for a data flow diagram.

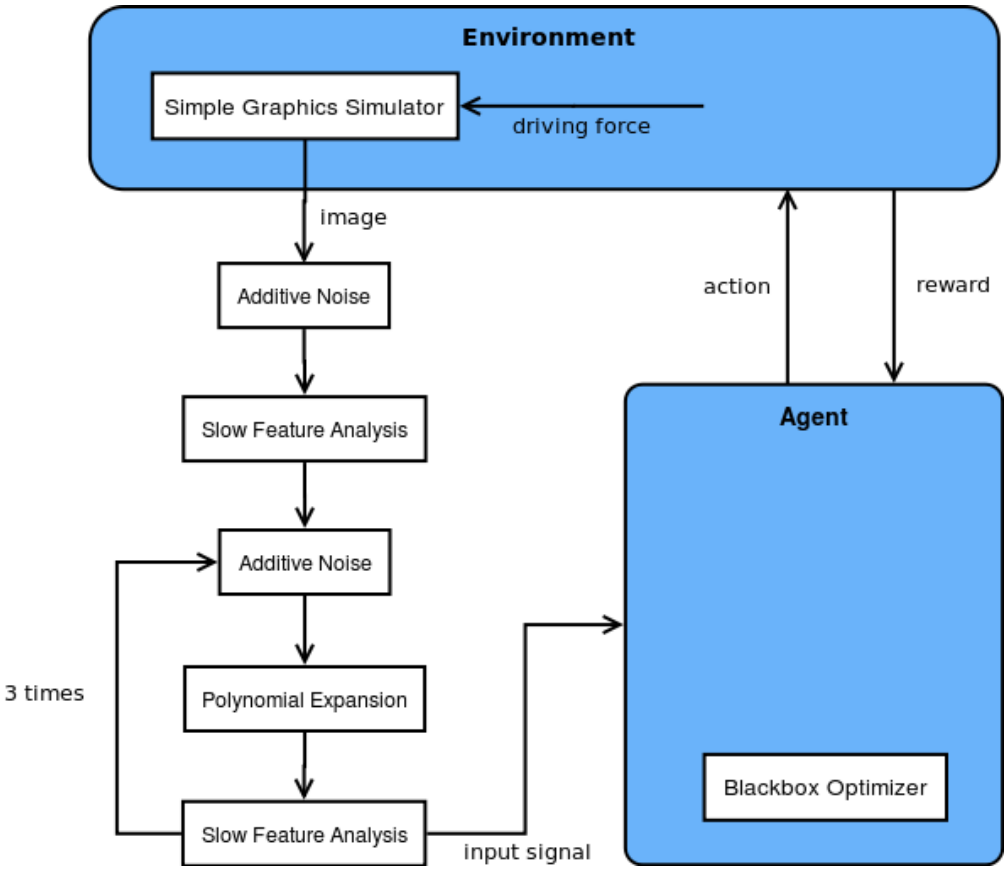


Figure 3.1: Data Flow Diagram

3.3 Libraries

We used the two major machine learning libraries PyBrain and MDP for this thesis. By using those two powerful libraries we were able to build a system which is easily adaptable and extendable. We will give a short overview over the two libraries.

3.3.1 PyBrain

PyBrain [SBW+10] stands for Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library. It is therefore a Toolbox providing algorithms from the above mentioned fields. It is very modular and can be easily applied and extended to perform machine learning. A PyBrain module consists of an input buffer, an output buffer and parameters to tune the underlying transformation algorithm. The transformation algorithm takes the input from the input buffer and applies an algorithm to create the output which will be present in the output buffer. A general module also provides error buffers. These are generally used for error back propagation. Modules can be stacked to create networks. A simple Artificial Neural Network can be created as easily

as stacking linear layers on top of softmax layers and so on. More complex networks are possible too. Any directed graph can be realized by connecting the input and output buffers of modules in an appropriate way. Any network is also a module, which makes it easy to create nested networks. PyBrain offers a variety of algorithms to train these modules. It provides out of the box algorithm for Supervised Learning, Unsupervised Learning, Reinforcement Learning, and Optimization Problems.

Reinforcement Learning using PyBrain

A general Reinforcement Learning task is split in three parts:

- **Environment**
- **Task**
- **Learner**

The **Environment** is a virtual world and does not know anything about tasks, learners or rewards. It changes from state to state as actions are applied to it. A **Task** defines the reward function, scales inputs and outputs and tells the learner when an episode ends. Many different tasks can be constructed for one Environment. The **Learner** will contain the learning algorithm and will use the rewards obtained from the task to adapt the algorithm's parameters.

PyBrain provides the following algorithms to be used in RL (List is taken from the official documentation):

- Value-based:
  - Q-Learning (with/without eligibility traces)
  - SARSA
  - Neural Fitted Q-iteration
- Policy Gradients:
  - REINFORCE
  - Natural Actor-Critic
- Exploration Methods:
  - Epsilon-Greedy Exploration (discrete)
  - Boltzmann Exploration (discrete)
  - Gaussian Exploration (continuous)
  - State-Dependent Exploration (continuous)

Black-box Optimization using PyBrain

Algorithms for black-box optimization can be applied to any problem which can be constructed as the minimization of an error function. A problem only needs to implement a fitness evaluator Interface to be used in black-box optimization. The fitness evaluator provides the black-box optimizer with a fitness function, which it has to maximize. In RL the fitness function is the cumulated reward gained in one episode. The optimization algorithm will change parameters of the network after every episode by evaluating the fitness function.

The following is a list of implemented algorithms which can used in black-box optimization (taken from the official documentation):

- (Stochastic) Hill-climbing
- Particle Swarm Optimization (PSO)
- Evolution Strategies (ES)
- Covariance Matrix Adaption ES (CMA-ES)
- Natural Evolution Strategies (NES)
- Fitness Expectation-Maximization (FEM)
- Finite Difference Gradient Descent
- Policy Gradients with Parameter Exploration (PGPE)
- Simultaneous Perturbation Stochastic Approximation (SPSA)
- Genetic Algorithms (GA)
- Competitive Co-Evolution
- (Inner/Inverse) Memetic Search
- Multi-Objective Optimization NSGA-II

Supervised Learning using PyBrain

In Supervised Learning we have the input samples and its desired output values. These training samples are used to train a classifier in case of a discrete output space or find a regression function in case the output space is continuous. The goal of a Supervised Learning algorithm is to find a function which generalizes the given data to make accurate predictions for future samples.

The following is a list of implemented algorithms, which can used out of the box with the PyBrain library:

- Back-Propagation
- R-Prop
- Support-Vector-Machines (LIBSVM interface)
- Evolino

Unsupervised Learning using PyBrain

The exact opposite of Supervised Learning is the field of Unsupervised Learning. Algorithms in this field are meant to find structure or other forms of 'meaning' in a given unlabeled data set. This is done by analyzing how data is organized and trying to explain key features in the data.

Unsupervised Learning algorithms in PyBrain include the following:

- K-Means Clustering
- PCA/pPCA
- LSH for Hamming and Euclidean Spaces
- Deep Belief Networks

3.3.2 MDP

MDP [ZWWB08] stands for Modular toolkit for Data Processing. It is a Python data processing framework. It includes many supervised and unsupervised data processing algorithms. All algorithms are packaged in modules and can be combined to build arbitrary feed forward networks. It is easily extensible and provides the researcher with pipeline topological structure for data processing. Special care has been taken to optimize the provided algorithms in terms of memory and speed. It provides means to adjust the precision of the internal calculations and to parallelize the computations.

A MDP network consists of connected nodes. A node is the basic building block of every network. Every node can have one or more training phases. MDP takes care of the training phases for all the nodes in the network in the right order. It automatically determines the input and output dimensions for every node, as data is provided during the training phases. The data can be treated in batch mode or in an online mode for amounts of data that do not fit in memory at one time. The online mode also provides the user with the possibility of generating the data on-the-fly.

After the training phases are over, the trained MDP network can be executed with new data. If all the nodes in one network use algorithm which provide an inversion, then the hole network can also be used upwards and invert the already processed data.

The MDP library is therefore a useful machine learning library capable of doing fast calculations on data in a feed forward way. Its pipeline architecture provides strong flexibility and gives a researcher means to perform many out-of-the-box algorithms on his data.

The number of algorithms implemented in the MDP library is steadily growing. The following is a a current list of implemented algorithms:

- Linear Regression
- Fisher Discriminant Analysis
- Locally Linear Embedding

- Hessian Locally Linear Embedding
- Restricted Boltzmann Machine
- Gaussian Classifiers
- Growing Neural Gas
- Factor Analysis
- K-Means Clustering
- PCA (standard, NIPALS)
- ICA (CuBICA, FastICA, TDSEP, JADE, XSFA)
- Slow Feature Analysis
- Independent Slow Feature Analysis
- Polynomial Expansion
- Hit Parades
- Noise
- Time Frames

3.4 Experiments

Two experiments were implemented to test the idea developed in this thesis. The first implementation is the well known cart-pole experiment. For the second task we chose the classic game of Pong.

3.4.1 Cart-Pole

The classic cart-pole control experiment is a balancing task. The goal of the task is to balance the pole on the cart without moving out of a defined area. The possible actions $a \in A$ are limited to moving the cart left and right. $A = \{left, right\}$. The pole is centered on the cart and attached via a passive joint.

The complete state is defined by the pole angle θ, the angle speed ω, the cart position x, and the cart's speed v. The driving force d_{cart} for this experiment:

$$d_{cart} = \{\theta, \omega, x, v\}$$

The driving force is initialized partly random.

$$\text{init}(d_{cart}) = \begin{cases} \theta \in [-0.2, 0.2] \\ \omega = 0 \\ x \in [-0.5, 0.5] \\ v = 0 \end{cases}$$

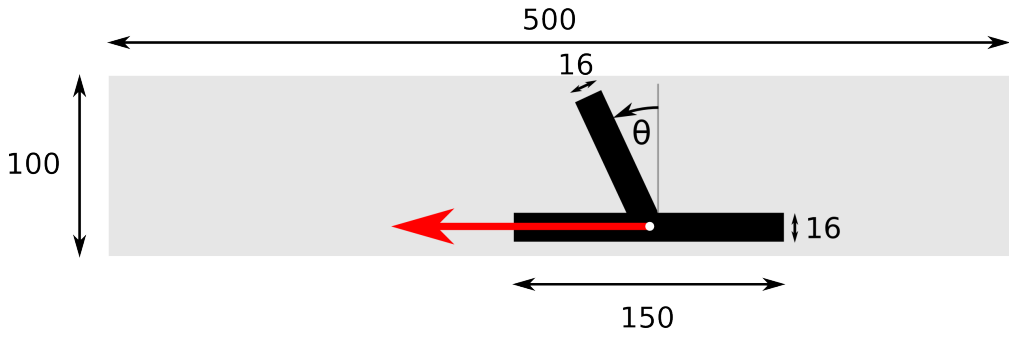


Figure 3.2: Cart pole Task Layout

For the RL we need to define a reward signal $r(t)$.

$$r(t) = \begin{cases} 0, & if\ |\theta| < 0.05 \text{ and } |x| < 0.05; \\ -2 * (\text{maxsteps} - n), & if\ |\theta| > 0.7 \text{ or } |x| > 2.4; \\ -1, & \text{otherwise.} \end{cases} \tag{3.2}$$

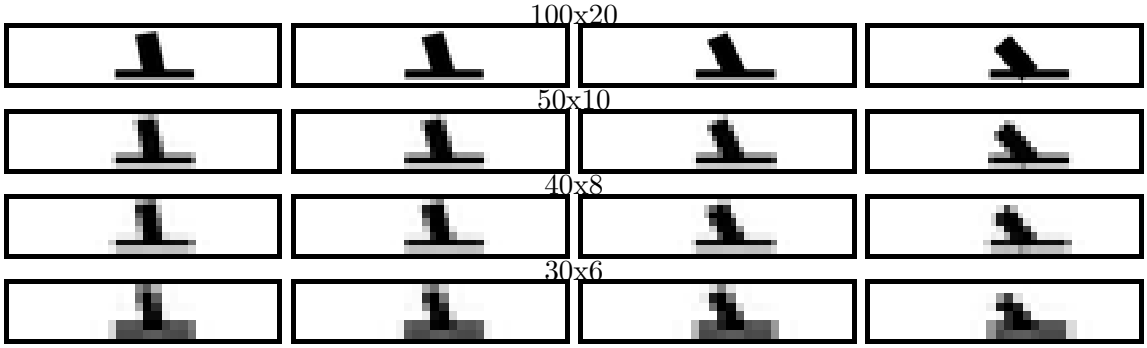
This means that a reward of 0 is given as long as the cart position x is very close to the start and the pole is nearly straight up. A reward of -1 is given when the cart is further away or the pole is not in a nearly straight position. A reward of -2 is given for all remaining time steps when the episode is canceled, because the pole has surpassed a curtain threshold or the cart position is too far out.

One episode for the cart pole experiment ends when one of the following conditions is met:

- $|\theta| > 0.7$
- $|x| > 2.4$
- $n >=$ maxsteps

The driving force d_{cart} was used to generate different sized images for comparison. See sample images in table 3.1.

Table 3.1: Comparing different image sizes for the Cart pole Task



3.4.2 Pong

The game Pong is one of the earliest arcade games. It is a tennis sport game first released in 1972. In this thesis we use an adapted version of Pong which is playable by one player. The second player is replaced by a wall. The reward is the number of times the agent is able to bounce the ball and continue to play. An episode ends as soon as the agent misses the ball or when he achieves a reward of 10, which equals 10 bounces. The total game-state is defined by the ball position, a vector describing its speed and direction and the paddle position. We will call these game-state parameters driving force d_{pong} from now on:

$$d_{pong} = \{ballpos, dir, paddlepos\}$$

The ball direction dir of the driving force d_{pong} is initialized randomly. A random angle $\alpha \in [-70°, 70°]$ is chosen and the vector dir is calculated from that. The fully initialized state of d_{pong} therefore is:

$$\text{init}(d_{pong}) = \begin{cases} ballpos = \begin{pmatrix} 50 \\ 50 \end{pmatrix} \\ dir = \left\| \begin{pmatrix} -1 \\ \tan(\alpha) \end{pmatrix} \right\| \\ paddlepos = 50 \end{cases}$$

Parameters which stayed constant through one test run, but were adapted to test the thesis idea include the following:

Parameter	Values	Explanation
Image Size	5x5, 10x10, 20x20, 50x50	in pixels
Paddle Speed	5, 15	in percent on the y axis per time-step

See chapter Results 4 for results of using different values for these parameters.

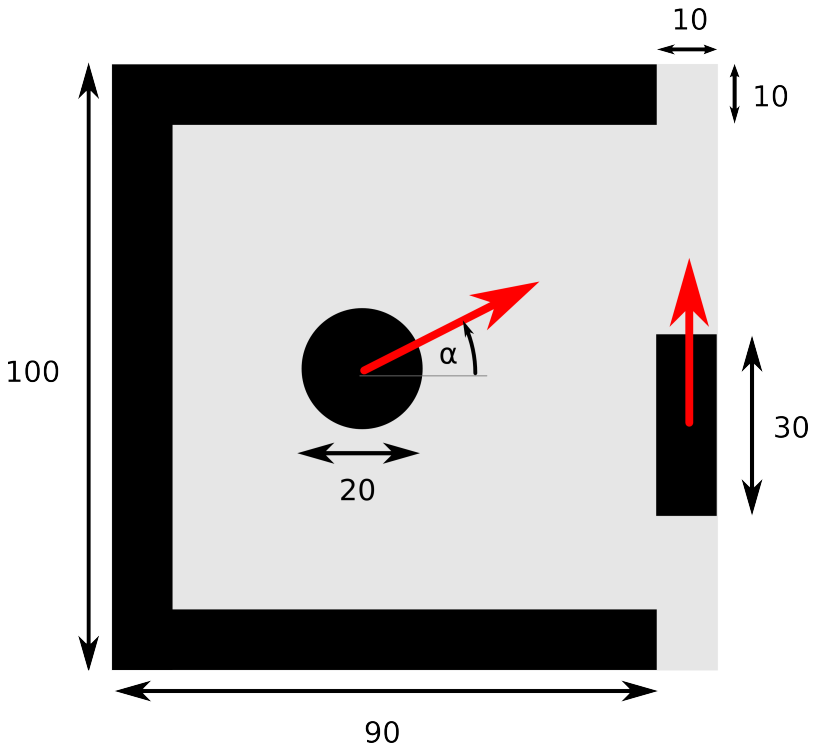


Figure 3.3: Pong Task Layout

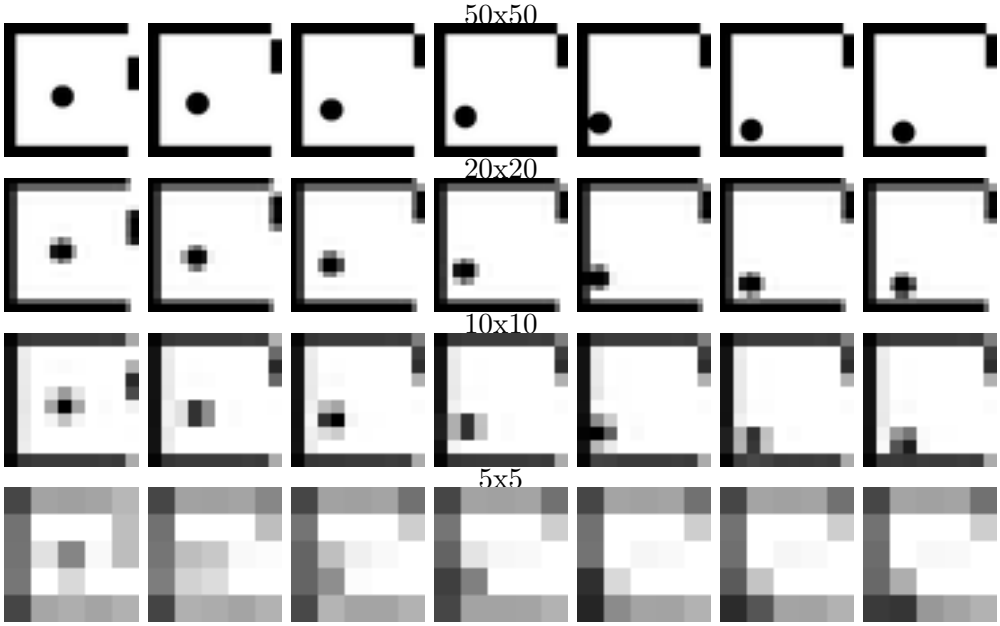
For the RL we need to define a reward signal $r(t)$:

$$r(t) = \begin{cases} 1, & \text{if bounce;} \\ 0, & \text{otherwise.} \end{cases}$$

This reward signal $r(t)$ will add up to a total Reward R_t with a max of 10 per episode when playing optimal.

In table 3.2 we can see the different image sizes being used for the training.

Table 3.2: Comparing different image sizes for the game Pong



Chapter 4

Results and Discussion

Overall we were able to show that Reinforcement Learning on slow features is an approach with promising results. In our experiment setup we used two different tasks to test the thesis idea. The cart pole experiment, being the first approach was of limited success. The SFA was only in parts successful in extracting useful features for the RL. The Pong experiment on the other hand was very successful. We can show that the features extracted by the SFA were highly correlated with the underlying driving force. Learning on those slow features yields better results than learning on the driving force itself. This advantage can not be explained by the SFA being better in removing the small amount of added noise to the data, as it is also better than the RL on the driving force without noise. It has to be assumed that a SFA preprocessing does not only extract useful features, but is also able to prepare features in a suitable way for the later RL. This was tested by using the SFA directly on the driving force, which yielded an even better result than the learning on the image data. It is known though that information is lost by creating images from the driving force.

In this section we want to describe the obtained results in detail for both experiment setups used. A correlation analysis is conducted and the overall results will be described.

4.1 Pong Experiment

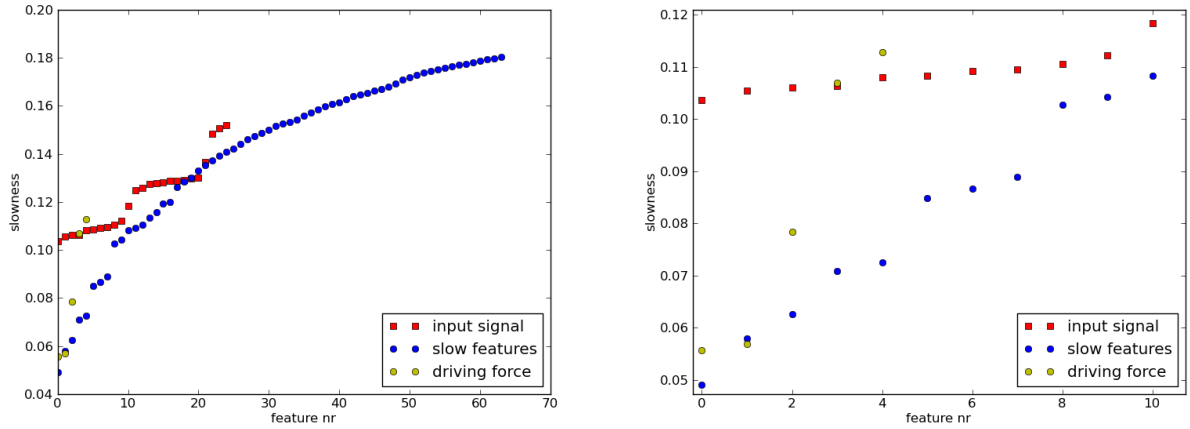
Please see details for the implementation in section 3.4.2.

Slowness Analysis

The correlation analysis will give us a good idea about how many and which slow features should be considered for the RL. For the general case, in which we don't have the driving force, we either have to estimate the number of useful slow features or we use the slowness of the extracted features. Instead of using the Δ directly we use the slowness measure proposed in [WS02]:

$$\eta(x) = \frac{T}{2\pi}\sqrt{\Delta(x)} \tag{4.1}$$

Figure 4.1: Slowness of the Pong task with size 5x5



(a) all features visible

(b) zoomed in

Please see section Measuring Slowness 2.2.2.3 for details.

We used two different experiment setups with the Pong task: Pongeasy and Ponghard. The two setups differ in the speed of the paddle. For Pongeasy the maximum speed value of the paddle was chosen to be faster than the one of the ball and for Ponghard the speed of the paddle was chosen to be slower. The results for both tasks were pretty similar and the rest of the discussion will focus on the setup setting Pongeasy. The reason for the similar results is the limited range for the angle of the ball which limits the chance that the ball will be faster than the paddle in y-direction in any case.

Please see figure 4.1, figure 4.2, figure 4.3 and figure 4.4 for a comparison of the slowness of all the input features and the found slow features with the SFA. For the sizes $5x5$, $10x10$, $20x20$ only the first 8-9 slow features are slower than the slowest features of the input signal, excluding constant input features. This is a good indicator for how many slow features will be useful for the RL. The $50x50$ input signal had a lot more constant features on the border of the image and 29 slow features were slower than the slowest, non-constant feature of the input space.

The slowness of the driving force is ordered in the following way.

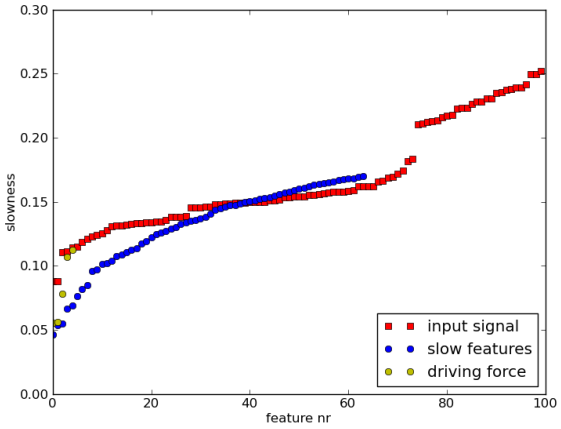
$$\eta(ballpos_y) < \eta(paddlepos) < \eta(ballpos_y) < \eta(balldir_y) < \eta(balldir_x)$$

For the second task we need to extract the speed and direction information of the paddle to find an optimal strategy π^*. To represent this information in the input signal we literally added one past frame to the current. This way the information for the speed and direction is encoded in the difference of the two provided images.

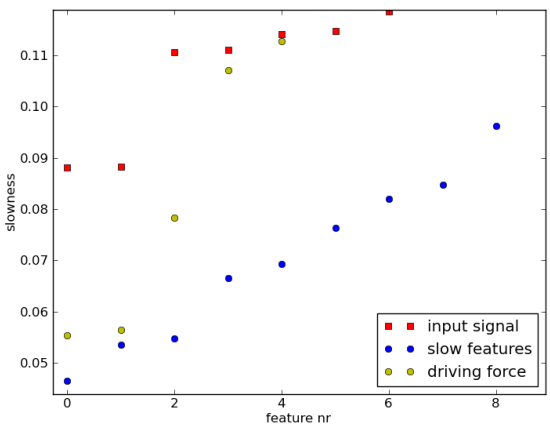
Please see figure 4.5, figure 4.6 and figure 4.7 for a comparison of $\eta(x)$ for the input signal, the driving force and the found slow features.

From the slowness of the driving force we can see why the SFA was only

Figure 4.2: Slowness of the Pong task with size 10x10

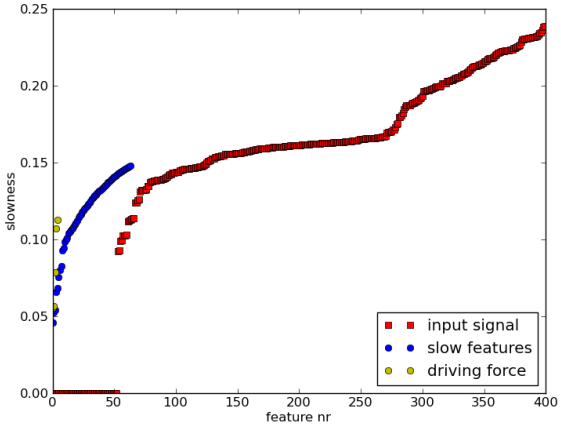


(a) all features visible

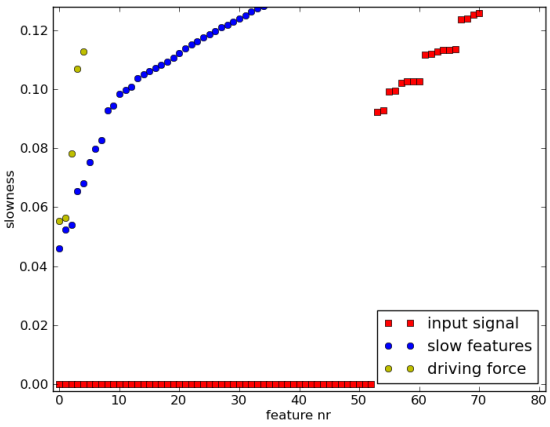


(b) zoomed in

Figure 4.3: Slowness of the Pong task with size 20x20

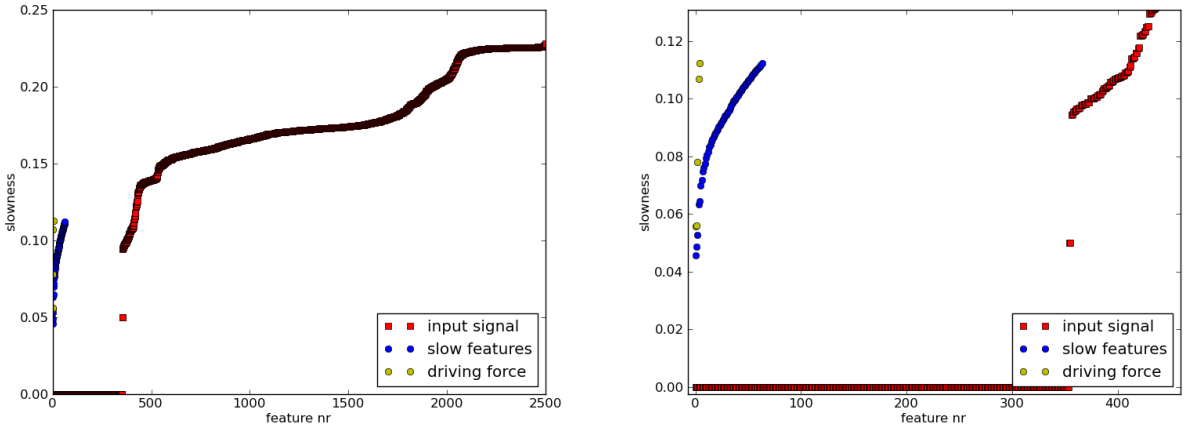


(a) all features visible



(b) zoomed in

Figure 4.4: Slowness of the Pong task with size 50x50



(a) all features visible

(b) zoomed in

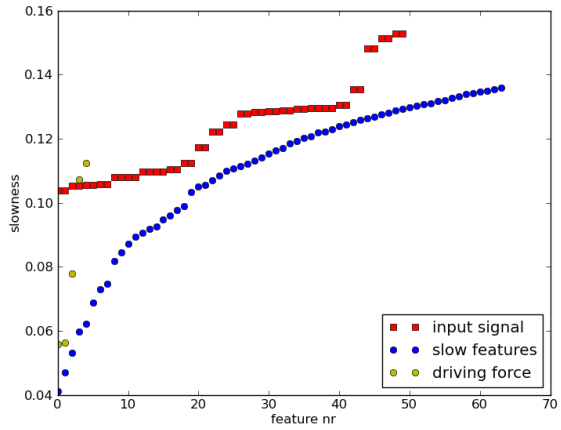
able to find the position of the ball and the paddle. The direction of the ball has a higher $\eta(x)$ than some of the input signal features. This is the reason why we do not find a a good match for that part of the driving force d_{pong}.

In the figures 4.8 we can see a heat map plot of the slowness for all input signals. It can be seen that for the bigger input space with the image size 20x20 and 50x50 the walls of the image never change and have a slowness $\eta(x) = 0$. The reason for the higher slowness of the wall pixels for the smaller images is the anti-aliasing effect .

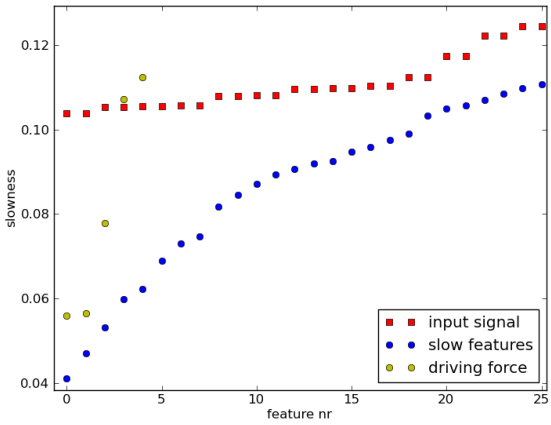
Correlation Analysis

Useful and significant features are needed for a successful RL. We use a Correlation Analysis to show that the SFA was able to find features which represent important information of the driving force d_{pong}. See table 4.1 and table 4.2 for a listing of the correlation coefficients. It can be seen that the parts of the driving force which resemble position of the ball and the paddle can be extracted by the SFA. The correlation for these values is on average at $\sim 90\%$. The correlation values of the direction in the case of not using a time-frame do not make sense as the required information is not encoded in the image and can therefore not be extracted. The correlation values rather resemble the correlation between the position and the direction. An increase in correlation when using the time-frame can be seen, but it's not enough to conclude that the direction was found.

Figure 4.5: Slowness of the Pong task with size 5x5 with time-frame

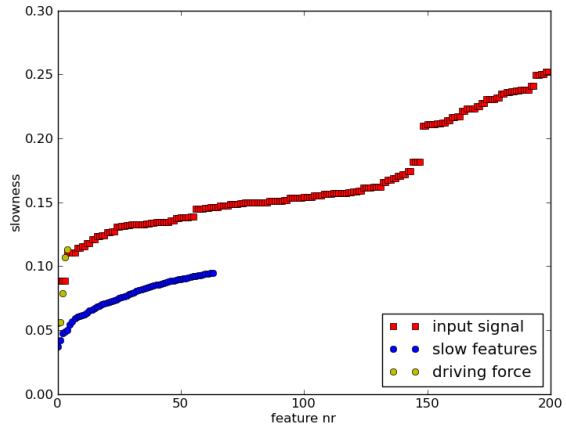


(a) all features visible

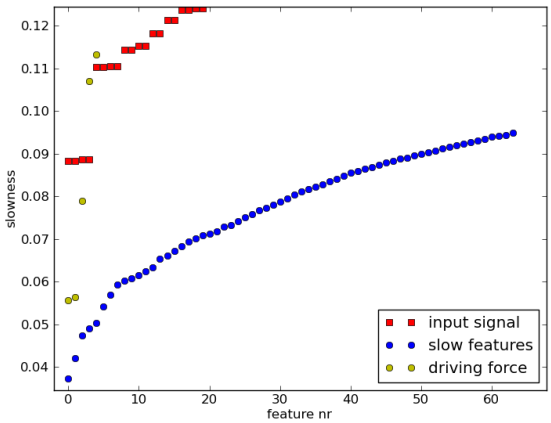


(b) zoomed in

Figure 4.6: Slowness of the Pong task with size 10x10 with time-frame



(a) all features visible

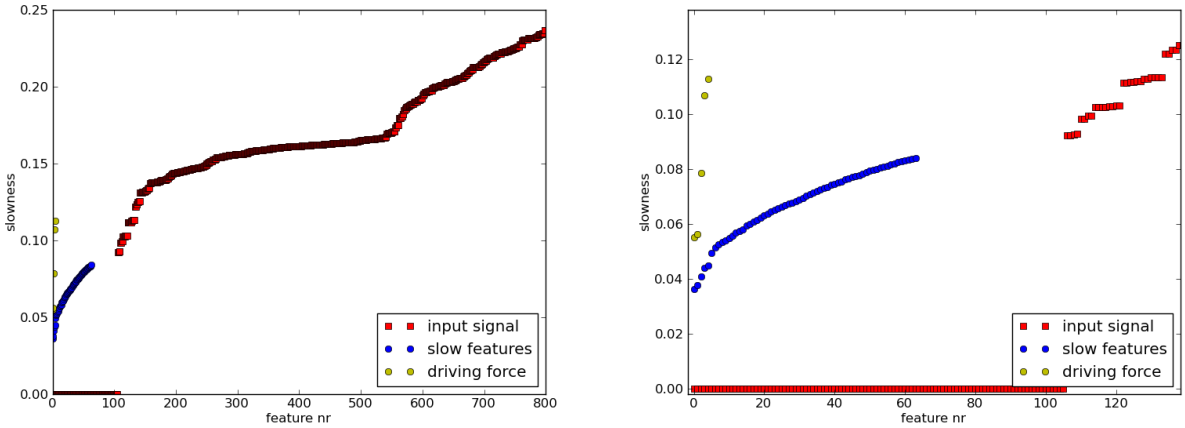


(b) zoomed in

Table 4.1: Correlation Coefficients woTF

Parameter	5x5	10x10	20x20	50x50	Slow Feature Nr
$ballpos_x$	0.8664	0.8823	0.8878	0.8933	2
$pallpos_y$	0.8884	0.9154	0.9347	0.9340	0
dir_x	0.2572	0.2405	0.2233	0.2204	2
dir_y	0.1675	0.1298	0.1075	0.1053	18
$paddlepos$	0.9072	0.9241	0.9309	0.9329	1

Figure 4.7: Slowness of the Pong task with size 20x20 with time-frame



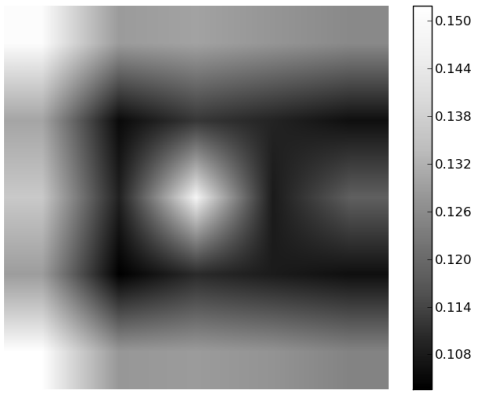
(a) all features visible

(b) zoomed in

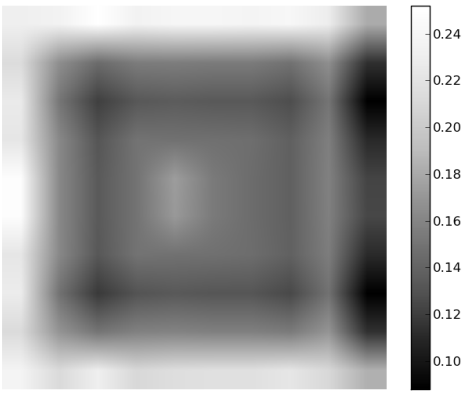
Table 4.2: Correlation Coefficients with time frame

Parameter	5x5	10x10	20x20	Slow Feature Nr
$ballpos_x$	0.9096	0.8601	0.7590	2
$pallpos_y$	0.9423	0.8963	0.7903	1
dir_x	0.1697	0.2839	0.2882	9
dir_y	0.2068	0.2855	0.3031	13
$paddlepos$	0.962	0.9595	0.9492	0

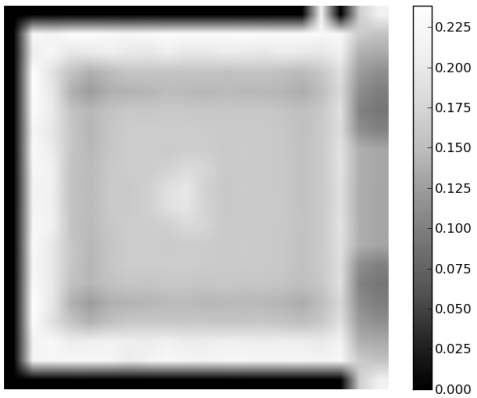
Figure 4.8: Slowness heat map for the different image sizes



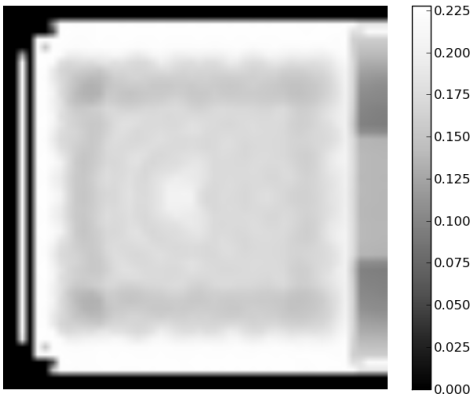
(a) 5x5



(b) 10x10



(c) 20x20



(d) 50x50

Results

The overall results for the learning system show that it can be used successfully for this task. Please see figure 4.9 for overall result comparison. In this plot three different settings were compared:

- RL on the bare driving force
- RL on PCA preprocessed image data
- RL on PCA and ICA preprocessed image data
- RL on SFA processed image data

The RL on SFA processed image data did not find an optimal strategy, but it was able to out-compete Principal Component Analysis (PCA) with RL and the RL on the bare driving force. It was also able to perform better than a system consisting of a layer of Independent Component Analysis (ICA) on top of a PCA layer as a preprocessing stage before applying RL. The plotted data is the averaged reward during the training phase over 100 runs, with each consisting of 5000 episodes. It is low pass filtered for better comparison.

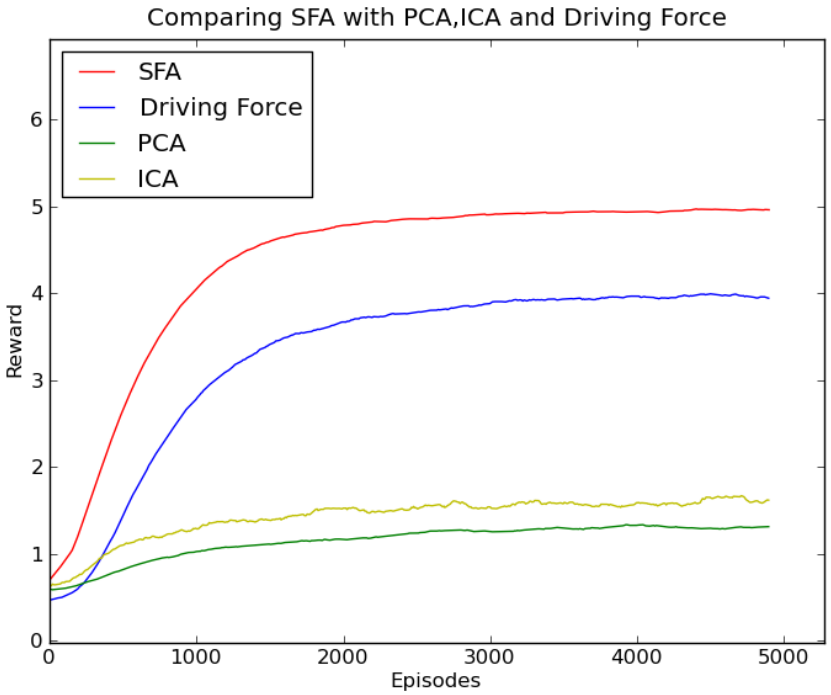


Figure 4.9: Pong Experiment Results

In a second figure 4.10 we can see a comparison how good the RL fared with the driving force. There is only a small advantage when using noise for generalization. Using the SFA as preprocessing for the bare driving force yielded

a very good learning. It can be seen that SFA preprocessing of the input state space is beneficial for the RL.

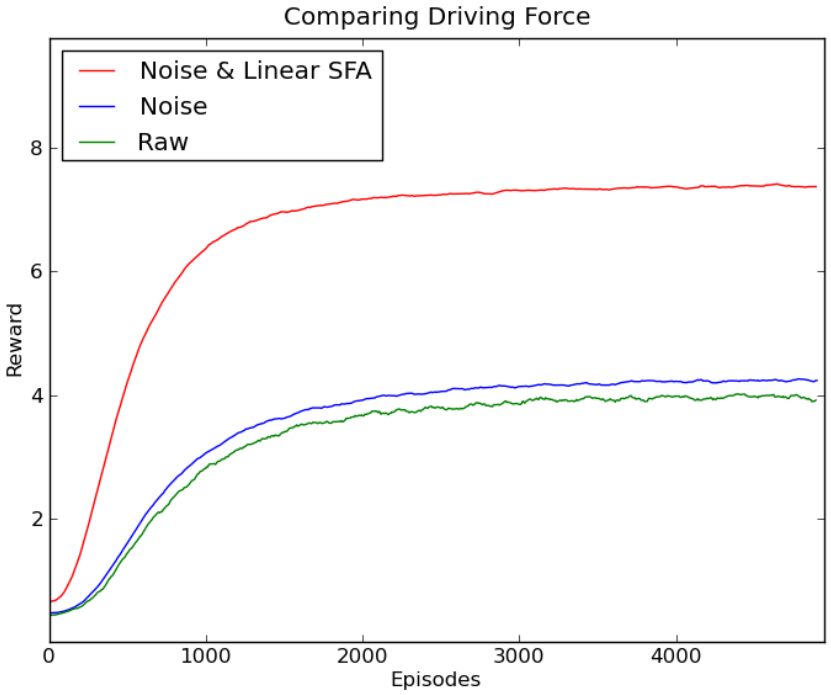


Figure 4.10: Comparing the Driving Force on the Pong task

4.2 Cart-pole Experiment

Please see section 3.4.1 for details of the implementation of this task.

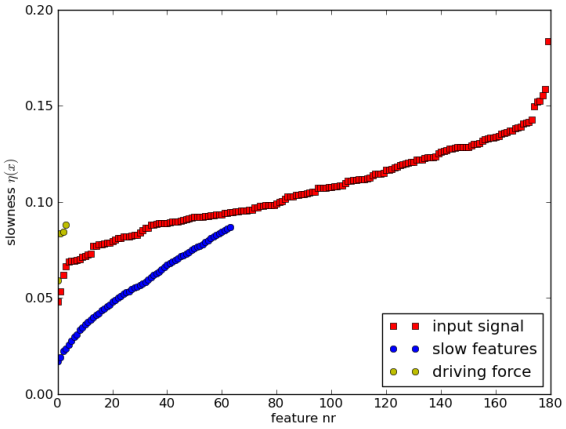
Slowness Analysis

The Cartpole Experiment's performance was dramatically worse when compared to the Pong Experiment. One explanation is the slowness $\eta(x)$ of the driving force. All parts of the driving force were faster than many features of the input signal. Therefore slowness is not a good indicator for finding these features. Please see the figures 4.11,4.12,4.13 and 4.14 for detailed charts of the slowness for the different image sizes. Most found slow features are a lot slower than the driving force but also the input signal. What kind of information is encoded in these features can only be speculated and is not known at this point.

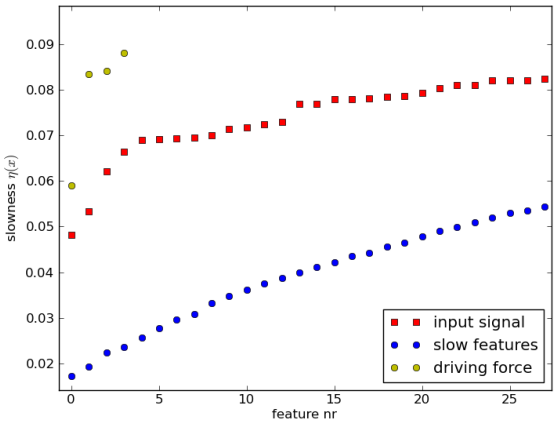
Correlation Analysis

For the correlation analysis we want to see which features of the driving force $d_{cart} = \{\theta, \omega, x, v\}$ can be found by the SFA. The found features correlate much

Figure 4.11: Slowness of the cartpole task with size 30x6

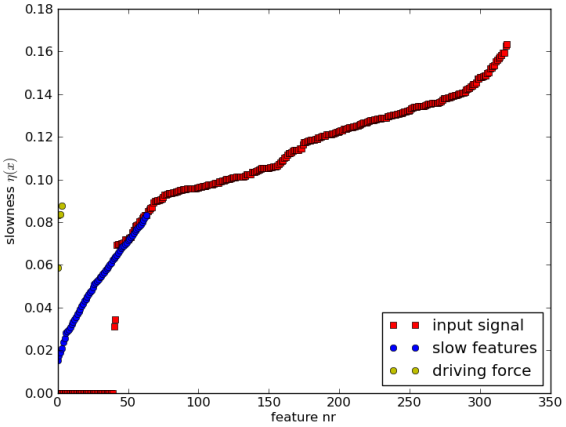


(a) all features visible

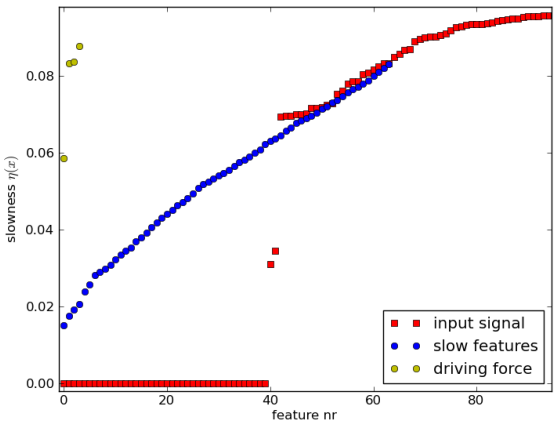


(b) zoomed in

Figure 4.12: Slowness of the cartpole task with size 40x8

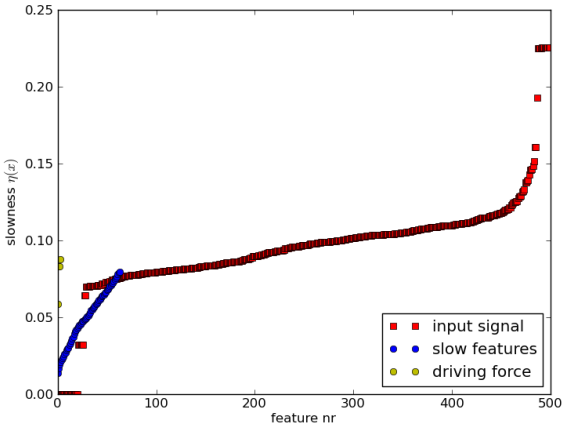


(a) all features visible

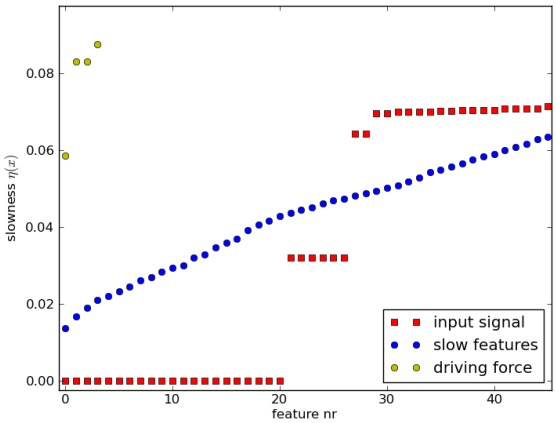


(b) zoomed in

Figure 4.13: Slowness of the cartpole task with size 50x10

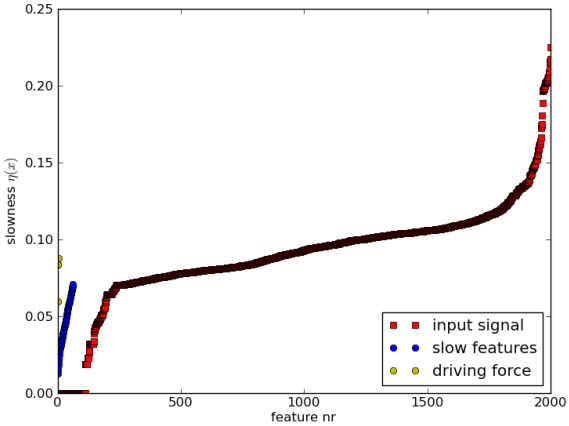


(a) all features visible

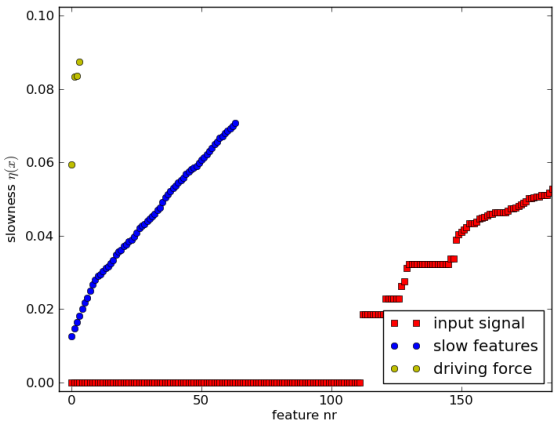


(b) zoomed in

Figure 4.14: Slowness of the cartpole task with size 100x20



(a) all features visible



(b) zoomed in

Table 4.3: Correlation Coefficients without time frame

Parameter	30x6	40x8	50x10	100x20	Slow Feature Nr
θ	0.5315	0.4747	0.3200	0.4312	28-50
ω	0.4050	0.3711	0.2612	0.3469	28-50
x	0.5093	0.4952	0.5112	0.3925	11-27
v	0.2785	0.2649	0.1961	0.2581	7-41

Table 4.4: Correlation Coefficients wtf

Parameter	30x6	40x8	50x10	Slow Feature Nr
θ	0.6014	0.5079	0.5305	25-34
ω	0.4565	0.3881	0.4141	25-34
x	0.4707	0.5279	0.4185	4-22
v	0.3172	0.2707	0.3039	25-34

worse than in the Pong Experiment. We gave a potential explanation in the Slowness Analysis section (see section 4.2). One thing we can see is that the found features do correlate to a certain degree, but not with the slowest features found. All driving force features correlate with rather fast slow features. Many slower features can be extracted, which do not correlate with the driving force.

It is clear that the information of the speed of the wagon and the angle is not encoded in one still image and therefore can not be extracted at all. The found correlation rather resembles a high correlation between the speed and position(~ 0.43) and θ and ω(~ 0.78). Only a very slight improvement can be observed when a second frame, called a timeframe, is literally added to the first as input for the SFA.

Please see chapter Future Work 6 for thoughts on how to improve this aspect.

Results

The overall performance for the Cartpole Task was bad when compared to Learning on the bare driving force. Please see figure 4.15 for a comparison. One reason can be seen in figure 4.16. The RL on the bare driving force is better than the RL on the SFA preprocessed driving force. This indicates that the information encoding in the driving force is very suitable for RL. The SFA used with the video signal has a much worse performance. It is better than a preprocessing stage consisting of PCA or a combination of PCA and ICA though.

4.3 Limitations

Three main limitations were encountered during the evaluation of the learning system. Scalability, the inability to adapt to a changing environment and the

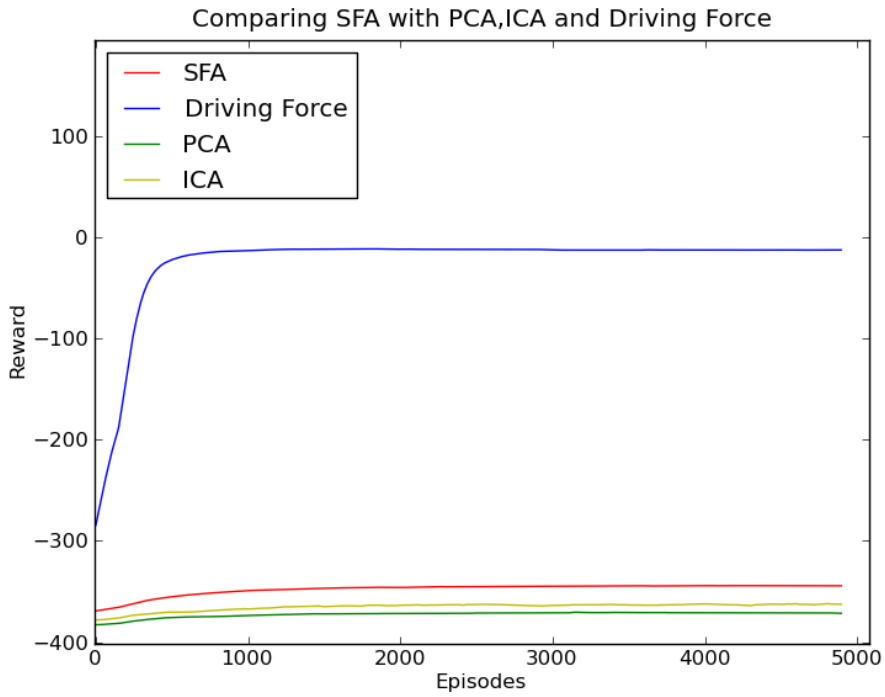


Figure 4.15: Comparing RL on the SFA processed Image Data with the Bare Driving Force on the Cart pole task

extraction of temporal information. Please see chapter Future Work 6 for ideas on how to improve on these limitations.

Scalability

The current learning system is limited in the size of the images it can process. The two control tasks Pong and Cartpole had input dimension of up to 2500 and 2000, respectively. The system can not handle a much bigger input space. The limitation lies in the SFA algorithm which suffers from the curse of dimensionality.

Adapting to non-stationary processes

The limitation of not being able to adapt to non-stationary processes is not relevant to the short lived control tasks used in this thesis. Different control tasks can thought up which will be limited by this effect.

Extraction of temporal information

The learning system implemented in this thesis was not able to extract temporal information from the input space. Both control tasks did not solely rely on

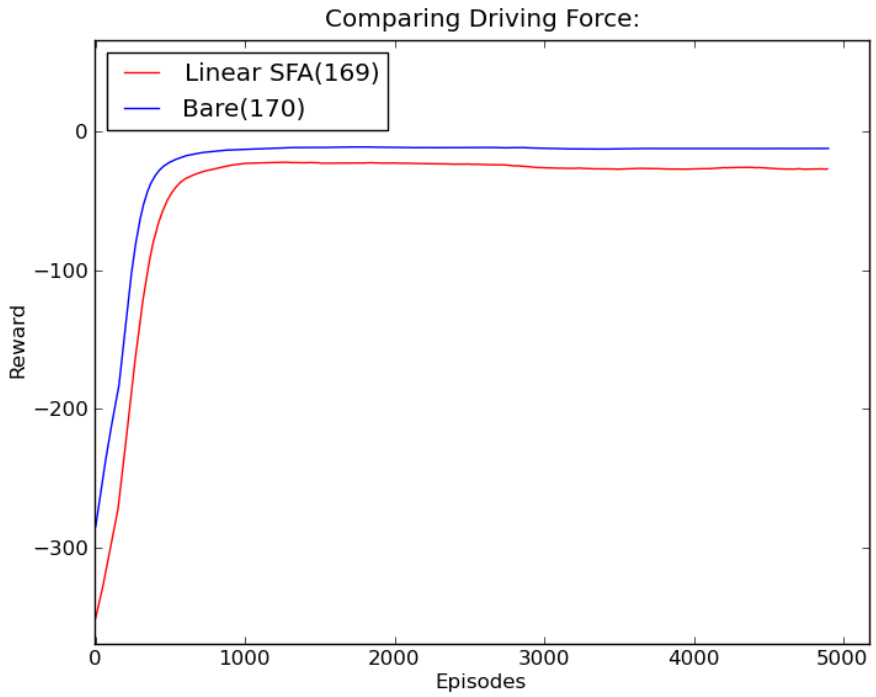


Figure 4.16: Comparing RL on the Driving Force and a SFA processed Driving Force on the Cart pole task

temporal information, but the extraction of this information would make it easier for the RL to learn optimal strategies π^*.

4.4 Comparison to other Feature Extraction Methods

We compared the unsupervised feature extraction method SFA to two other methods, PCA, and PCA together with ICA. We tested the system by swapping the SFA part of the algorithm with a PCA layer and a combined PCA and ICA layer. The combined layer of PCA and ICA fared better than PCA on its own. Both systems compared rather poor to the proposed learning system. It can be concluded that statistical independence and uncorrelation are not as good principles on its own as slowness is. Both PCA and ICA can supplement other methods to improve them. Please see figure 4.9 and figure 4.15 for a comparison on how they fared with the two control tasks Pong and the cart-pole task.

Chapter 5

Related Work

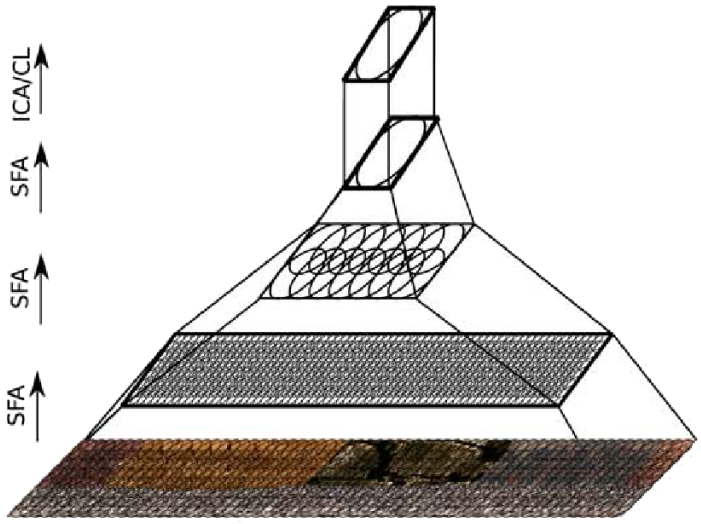
In this chapter we want to briefly summarize what research has been done on topics relevant to this thesis. The theory of SFA has been explained in detail in chapter Theoretical Foundation 2. For further information on SFA itself please consult the following publications: [Wis98, WS02, BW03, Wis03, SW08]. This chapter includes related work, which uses other feature extraction methods with RL and also other uses of SFA and how SFA can be extended. The chapter 6 Future Work will refer to some of the extensions of SFA discussed in here.

5.1 Receptive Fields

The idea to use receptive fields with SFA is to combat the curse of dimensionality. Any task that requires the SFA to find higher order features than just linear features will have to use some kind of polynomial expansion of the input space. The polynomial expansion of big input spaces like pictures or video signal will lead to unfeasible calculations in the SFA algorithm. The use of receptive fields is proposed for this reason [FSW07]. It is proposed to use the SFA on small overlapping segments of the input image instead of applying the SFA on the hole picture at once. Polynomial expanding small image segments in the order of 10x10 pixel will avoid the curse of dimensionality. A hierarchical architecture is then used to extract higher order slow features on every layer by applying SFA to the underlying receptive fields. An example can be seen in figure 5.1. The last layer will then apply SFA for the last time on the already found slow features. They were able to show that the use of SFA in the proposed way and an additional sparse coding step can lead to place, head-direction, and spatial-view cells.

Further research has been done by [LWW10]. They propose a similar system of using RL on top of the extracted slow features as it is being proposed in this thesis. In their system they use the above mentioned hierarchical architecture to extract slow features. A subsequent neural network is then trained by a reward-based synaptic learning rule which can be compared to Q-learning. The hole system was tested on the Morris water maze task. They were able to show that the speed and success of this learning system is comparable to that found in experimental studies with rats and therefore support the hypothesis that

Figure 5.1: An example for the use of receptive fields with Slow Feature Analysis [WS02]. The architecture consists of three layers of SFA and a final layer performing ICA for sparse coding the output.



slowness learning is one important unsupervised learning principle being used in the brain.

5.2 Kernelized Slow Feature Analysis

The curse of dimensionality can also be solved with a different approach. The use of the kernel trick with SFA [BM02] can completely avoid the dimensionality problem. They propose the use of a similar objective function to extract slow features for which the kernel trick can be applied. With this trick they avoid the need to explicitly represent the polynomial expanded input space and can perform SFA on huge input spaces.

The objective function used tries to maximize output variance over long time periods while trying to minimize output variance over short time periods. The use of the kernel trick allows the projection of the input space into a non linear kernel induced feature space without the need of representing this high dimensional feature space. The objective function is biological plausible as it can be implemented by a mixture of Hebbian and anti-Hebbian Learning on the same synapse. The resulting algorithm with the kernel trick can then be implemented by Sigma-Pi neurons or fixed radial basis function (RBF) networks, which gives the overall architecture a biological plausibility.

5.3 Online Slow Feature Analysis

The SFA algorithm used in this paper has a disadvantage when the task is non-stationary. After the training phase has finished it is not possible to adapt to changes in the environment. The use of SFA in the form of a neural network makes the algorithm more biological plausible and adds an online adaptivity, which is necessary for handling non-stationary processes.

SFA can be related to two standard neural architectures [WS02]. See figure 5.2 for the two neural structures. The non-linear expansion, here represented as non-linear basis functions $\tilde{h_k}(\mathbf{x})$, can be represented as synaptic clusters on the dendritic tree. They locally perform a fixed non-linear expansion on the input features. The output can be weighted independently to other non-linear expanding synaptic clusters. Sigma-pi units are examples, which can be used for this purpose.

A second plausible network provides the non-linear expansion through fixed non-linear units in a hidden layer in the network. The output layer consists of linear combinations of these non-linear features with trainable weights. A fixed RBF network is an example of this interpretation. Depending on the type of linear expansion needed, one or the other neural network is more suitable. In both cases the components of the input output function are given by:

$$g_j(\mathbf{x}) = \mathbf{w}_j^T \mathbf{h}(\mathbf{x}) = \tilde{w}_{j0} + \tilde{\mathbf{w}}_j^T \tilde{\mathbf{h}}(\mathbf{x})$$

The input signal $\mathbf{x}$ is assumed to be normalized and $\tilde{\mathbf{w}}_j$ is the appropriate raw weight vector.

The final part of the network consists of connections between the output units g_j and some variant of anti-Hebbian learning, which is needed to decorrelate the output. Both network architectures perform SFA and are to be seen as basic building blocks for a more complex hierarchical SFA architecture.

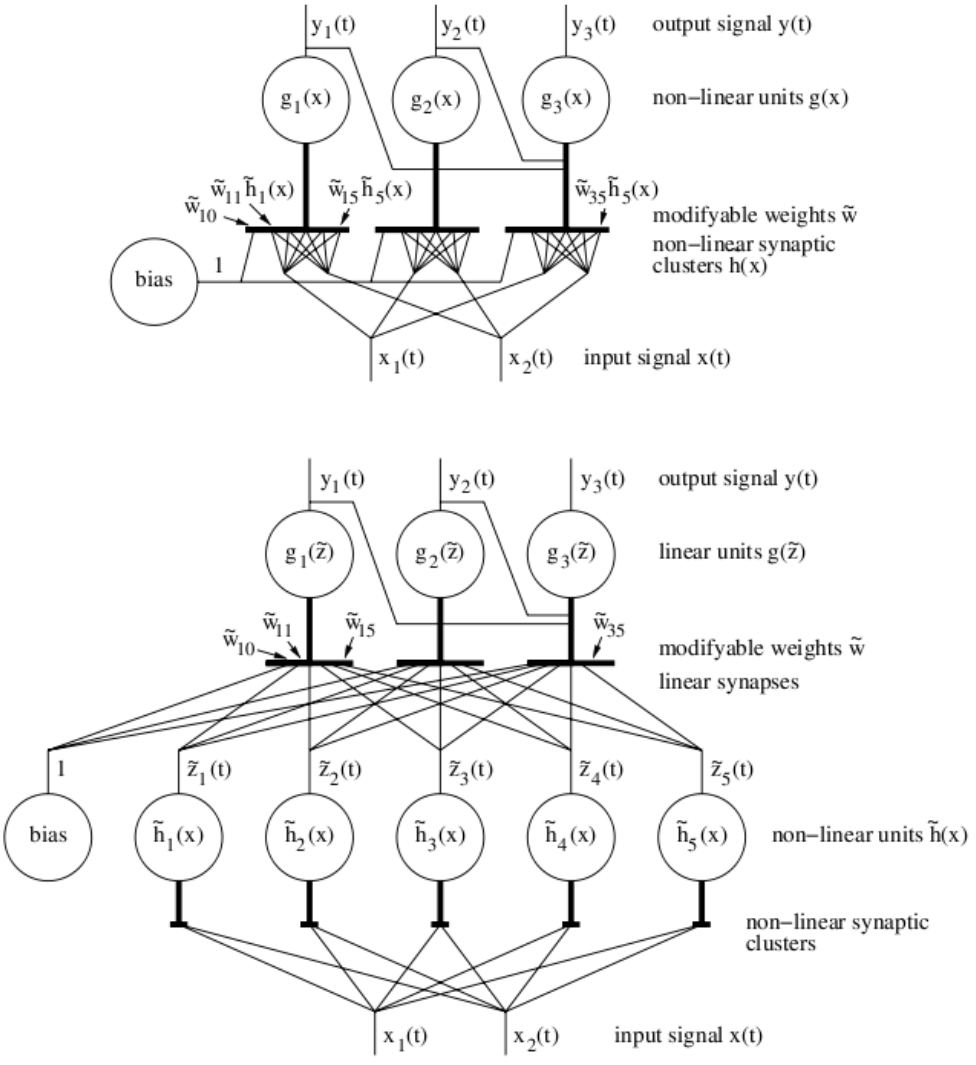
5.4 Reservoir Computing with Slow Feature Analysis

The use of temporal information is needed for many control tasks working with video signals. The use of time frames can add this temporal information to the input space. This means that the input consists of a vector of two consecutive image frames. This approach was used in [BW03]. They were able to extract the temporal information with SFA in this way.

A different approach is proposed by [AS09]. They propose the use of a reservoir with SFA. Reservoir Computing is a recently introduced paradigm in Recurrent Neural Networks (RNN). A reservoir is a randomly generated dynamic system. Only the output weights are trained while the internal weights in the reservoir stay fixed. Liquid-state machines and echo state networks are two major types in Reservoir Computing.

Using a reservoir with SFA adds short term memory to the system. This short term memory is represented by echoes, which are formed because of the recurrent connections within the reservoir. A later stage of SFA on top of

Figure 5.2: Two possible network architectures for SFA(Taken from [WS02]).



a reservoir can extract features which also contain the temporal information provided by the reservoir. The architecture proposed in [AS09] used a third layer using ICA for sparse coding the output. The system was used with a simulated robot to successfully self-locate in an environment by using 17 noisy short-range distance sensors.

5.5 Using PCA and ICA for feature extraction

SFA has been compared mathematically to ICA. The comparison of SFA and ICA came up with the result that linear SFA is formally equivalent to second order ICA with time delay of one [BBW06] while the conceptual differences between SFA and ICA will lead to very different results in the non-linear case. There work has been extended to a combined algorithm of SFA and ICA, named Independent Slow Feature Analysis [BZW07]. This new method was used for nonlinear blind source separation. The use of this algorithm for feature extraction in high dimensional spaces has not been tested yet.

PCA and ICA on the other hand have been used extensively for this purpose in different fields, like feature extraction for face recognition [BMS02, BLS98, HHH03, KKHK04]. These papers propose ICA as an alternative to the well researched use of PCA in the field of face recognition. The use of eigenfaces [TP91], holons [CM90] or local feature analysis [PA96] are successful examples for PCA in the field of face recognition. For a comparison of PCA and ICA on the FERET dataset see [BDBS02].

5.6 Using Gaussian Processes for dimension reduction

An alternative to using PCA and ICA is the use of Gaussian Processes [RW05] for feature extraction. The Gaussian Process Latent Variable Model [Law04, Law05] uses an adapted version of PCA to extract non-linear features. It can be used for dimension reduction and has been used in a learning system together with RL in [BHV10] for learning a bimanual reaching task. A similar approach using Gaussian Processes with RL is used in [MA09] to approximate a Poincaré map for learning biped locomotion. Both papers were able to show that the use of Gaussian Processes for feature extraction does work well in conjunction with RL.

Chapter 6

Future Work

There are many possibilities to further improve the proposed learning system. We can split those into two categories. Algorithmic enhancements and field of application. Algorithmic enhancement include all future work which is meant to improve the learning system. In this field we want to talk about improving scalability, adding adaptability for non-stationary process and the use of Reservoir Computing for adding temporal information. The second part field of application will include ideas about how and where we can apply and use the developed learning system.

6.1 Algorithmic Enhancements

Research in the field of SFA provides us with ideas on how to further improve the proposed learning system. Please see chapter Related Work 5 for details. First we want to solve the encountered limitations.

One limitation is scalability. Our system fairs well with the used video signal in the two experiment setups. So far the image size of the videos can not be increased much further, because of the curse of dimensionality affecting the SFA. A straight forward and easy to implement approach is the use of receptive fields [FSW07]. Another interesting approach is the use of the kernel trick. The use of the kernel trick avoids the curse of dimensionality by avoiding the need to represent the polynomial expanded input feature space explicitly [BM02].

The second limitation is about extracting temporal information. The used time frame, which is a simple concatenating of two consecutive images, was unsuccessful. An alternative idea to provide the relevant temporal information is the use of a reservoir [AS09]. Using an echo state network or a liquid state machine with SFA is an interesting concept and should be pursued further.

Another limitations which we found is the inability of the system to adapt to non-stationary processes. Any real world application will be confronted with some kind of changes in its environment. The use of an online version of SFA can speed up the learning process by instantly providing(at first inaccurate) slow features without the need of an extra SFA training phase. Additionally can a learning system based on an online version of SFA adapt to changes in its environment. Online SFA can be implemented using a neural architecture using

Sigma-pi units or fixed RBF networks as its basis for non linearly expanding the input space. The needed decorrelation of the output is then performed by an inter-connected layer of neurons using anti-Hebbian learning.

Further research should also be done in experimenting with different RL algorithms. The proposed learning system uses Fitness Expectation Maximization. A similar learning system proposed in [LWW10] uses a neural implementation of the Q-Learning algorithm. Many more RL algorithms are usable in this context and should be considered for further implementations of a similar learning system.

6.2 Ideas for Area of Application

The proposed learning system was tested on two different simulated control tasks. An interesting field of application is to go deeper into the field of simulated, but also real world robotics. The idea to learn motion primitives and trajectories with the system would be a challenging tasks. Until now it is not completely clear which control tasks are suitable for the learning system and which criteria are important to find relevant features for the RL stage. Some ideas were given on how to decide on this for certain control tasks. A detailed analysis about the encoding of important features in images and its speed η would make this clearer.

Chapter 7

Conclusion

Slow Feature Analysis is recent contribution in the field of unsupervised feature extraction. This field is very relevant for finding biological plausible algorithms as any creature on this planet is provided with a vast amount of information through its senses, which it needs to preprocess to perform any meaningful learning. Biological plausibility can provide good ideas for developing learning algorithms as the cognitive abilities of most animals outperform current robotic cognition by far. The proposed learning system is in its implementation not completely biological plausible, but equivalent biological plausible implementations exist.

It was our goal to test the usefulness of Slow Feature Analysis with Reinforcement Learning on complicated control tasks with huge input spaces. RL is a major field in machine learning and provides many means for learning in control tasks, but its algorithms require a small input space to perform useful learning in an control task. Therefore a preprocessing stage is needed for the RL. We used SFA to provide this preprocessing.

By testing the proposed learning system on two implemented control tasks we were able to find important task properties, which determine the success of RL on the extracted slow features. Tasks for which the assumption that slowness is an important indicator for the relevance of features holds, will perform very well with RL. On the other hand this is not generally true and tasks exist where this assumption does not hold. Overall we were able to show that slowness is often an important principle and that it can be used to perform Reinforcement Learning on huge input spaces.

Bibliography

[API] Amir Atiya, A.G. Parlos, and Lester Ingber. A reinforcement learning method based on adaptive simulated annealing. Lester ingber papers, Lester Ingber.

[ARL+09] R. M. Cohen Et Al, M. Romanski, J. E. Ledoux, J. Neurosci, J. L. Armony, J. D. Cohen, D. Servan-schreiber, K. P. Corodimas, J. E. Ledoux, Behav Neurosci, M. J. D. Miserendino, C. B. Sananes, K. R. Melia, M. E. P. Seligman, J. Abnorm Psychol, Wolfram Schultz, Peter Dayan, and P. Read Montague. A neural substrate of prediction and reward. 2009.

[AS09] Eric A. Antonelo and Benjamin Schrauwen. Unsupervised learning in reservoir computing: Modeling hippocampal place cells for small mobile robots. In *ICANN '09: Proceedings of the 19th International Conference on Artificial Neural Networks*, pages 747–756, Berlin, Heidelberg, 2009. Springer-Verlag.

[BA08] Joschka Boedecker and Minoru Asada. Simspark - concepts and application in the robocup 3d soccer simulation league. In *Proceedings of the SIMPAR-2008 Workshop: The Universe of RoboCup Simulators*, 2008.

[BBW06] T. Blaschke, P. Berkes, and L. Wiskott. What is the relationship between slow feature analysis and independent component analysis? *Neural Computation*, 18(10):2495–2508, 2006.

[BDBS02] Kyungim Baek, Bruce A. Draper, J. Ross Beveridge, and Kai She. Pca vs. ica: a comparison on the feret data set. In *in Proc. of the 4th International Conference on Computer Vision, ICCV'02*, pages 824–827, 2002.

[BHV10] Sebastian Bitzer, Matthew Howard, and Sethu Vijayakumar. Using dimensionality reduction to exploit constraints in reinforcement learning. In *IROS*, 2010.

[BLS98] Marian Stewart Bartlett, H. Martin Lades, and Terrence J. Sejnowski. Independent component representations for face recognition, 1998.

[BM02] Alistair Bray and Dominique Martinez. Kernel-based extraction of slow features: Complex cells learn disparity and translation invariance from natural images. In *Neural Information Processing Systems - NIPS'02*, page 8 p, Vancouver, Canada, 2002. none. Colloque avec actes et comité de lecture. internationale.

[BMS02] M. S. Bartlett, J. R. Movellan, and T. J. Sejnowski. Face recognition by independent component analysis. *IEEE Transactions on Neural Networks*, 13(6):1450–1464, November 2002.

[BW03] Pietro Berkes and Laurenz Wiskott. Slow feature analysis yields a rich repertoire of complex cell properties, January 2003.

[BZW07] Tobias Blaschke, Tiziano Zito, and Laurenz Wiskott. Independent Slow Feature Analysis and Nonlinear Blind Source Separation. *Neural Computation*, 19(4):994–1021, 2007.

[CM90] Garrison W. Cottrell and Janet Metcalfe. Empath: face, emotion, and gender recognition using holons. In *Proceedings of the 1990 conference on Advances in neural information processing systems 3*, NIPS-3, pages 564–571, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[FSW07] Mathias Franzius, Henning Sprekeler, and Laurenz Wiskott. Slowness and sparseness lead to place, head-direction, and spatial-view cells. *PLoS Computational Biology*, 3(8):e166, August 2007. (doi:10.1371/journal.pcbi.0030166).

[HHH03] A. Hyvarinen, P. Hoyer, and J. Hurri. Extensions of ica as models of natural images and visual processing. 2003.

[KKHK04] T.K. Kim, H.W. Kim, W.J. Hwang, and J.V. Kittler. Independent component analysis in a local facial residue space for face recognition. 37(9):1873–1885, September 2004.

[KLM96] Kaelbling, Leslie P., Littman, Michael L., and Moore, Andrew W. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Law04] Neil D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[Law05] Neil Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *J. Mach. Learn. Res.*, 6:1783–1816, December 2005.

[LWW10] Robert Legenstein, Niko Wilbert, and Laurenz Wiskott. Reinforcement learning on slow features of high-dimensional input streams. *PLoS Comput Biol*, 6(8):e1000894, 08 2010.

[MA09] Jun Morimoto and Christopher G. Atkeson. Nonparametric representation of an approximated poincaré map for learning biped locomotion. *Auton. Robots*, 27(2):131–144, 2009.

[MRG03] Shie Mannor, Reuven Rubinstein, and Yohai Gat. The cross entropy method for fast policy search. In *In International Conference on Machine Learning*, pages 512–519. Morgan Kaufmann, 2003.

[MSG99] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research*, 11:199–229, 1999.

[OORR04] Oliver Obst, Oliver Obst, Markus Rollmann, and Markus Rollmann. Spark - a generic simulator for physical multi-agent simulations. In *Computer Systems Science and Engineering*, 2004.

[PA96] Penio S. Penev and Joseph J. Atick. Local feature analysis: A general statistical theory for object representation, 1996.

[RW05] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, December 2005.

[SB99] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction by richard s. sutton and andrew g. barto, adaptive computation and machine learning series, mit press (bradford book), cambridge, mass., 1998, xviii + 322 pp, isbn 0-262-19398-1, (hardback). *Robotica*, 17(2):229–235, 1999.

[SBW+10] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 2010.

[SW08] Henning Sprekeler and Dr. Laurenz Wiskott. Understanding slow feature analysis: A mathematical framework, August 2008.

[TP91] M. Turk and A. Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[WB03] L. Wiskott and P. Berkes. Is slowness a learning principle of the visual cortex? *Zoology*, 106(4):373–382, December 2003.

[Wis98] Laurenz Wiskott. Learning Invariance Manifolds. In *Proc. of the 5th Joint Symp. on Neural Computation, May 16, San Diego, CA*, volume 8, pages 196–203, San Diego, CA, 1998. Univ. of California.

[Wis03] Laurenz Wiskott. Slow feature analysis: a theoretical analysis of optimal free responses. *Neural Comput.*, 15(9):2147–2177, 2003.

[WS02] Laurenz Wiskott and Terrence J. Sejnowski. Slow feature analysis: unsupervised learning of invariances. *Neural Comput.*, 14(4):715–770, 2002.

[WSPS08] Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. Fitness expectation maximization. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature*, pages 337–346, Berlin, Heidelberg, 2008. Springer-Verlag.

[ZWWB08] T. Zito, N. Wilbert, L. Wiskott, and P. Berkes. Modular toolkit for Data Processing (MDP): a Python data processing framework. *Frontiers in Neuroinformatics*, 2(8), 2008. (doi:10.3389/neuro.11.008.2008).