UNIVERSITÄT OSNABRÜCK

Fachbereich Humanwissenschaften,
Institut für Kognitionswissenschaft

Unsupervised Extraction of Visual Features for Neural Net-Based Reinforcement Learning

Bachelor Thesis

Written by:	Hannes Schulz
Supervisors:	Prof. Dr. Martin Riedmiller
	Prof. Dr. Peter König

Abstract

Visual information is easy to acquire but needs tedious preprocessing before it can be put to use to guide the learning of a task. In this thesis, we examine a framework where first, features optimized for stability and decorrelation over time are generated with an unsupervised learning technique for a simulated pendulum swing-up reinforcement learning task. Second, the fitness of these features for a neural net-based task learning algorithm is tested. We find that the task can be learnt on the features within a similar timescale as the direct learning on the simulated sensor data. Furthermore, we find that the stability of the features affects the performance of the task learner.

Contents

1. Introduction **3**
1.1. Learning from Visual Information 3
1.2. Aim of this Thesis 4

2. Methods **5**
2.1. An Unsupervised Network based on an Objective Function 5
2.2. Reinforcement Learning 6

3. An Architecture for Visual Task Learning **8**

4. Results **9**
4.1. Description of the Stimulus 10
4.2. Feature Generation in the Unsupervised Network 10
4.3. Reinforcement Learning Using Generated Features 11
4.4. The Impact of Stability on Performance 15
4.5. Effect of the Discretized Image 17

5. Discussion **18**

6. Conclusion **20**

References **21**

A. Appendix **23**

Ehrenwörtliche Erklärung **25**

1. Introduction

1.1. Learning from Visual Information

In physical systems, visually acquired information can be a cheap and easy substitute for specific sensor information. As a drawback, images usually contain much more information than necessary, posing a problem for their efficient utilization. A common way to deal with this problem is by filtering the images in such a way that only relevant information is retained. These filters are usually handcrafted or very general, like edge detectors and thus not problem specific.

In this work, we consider a method of feature generation without manual intervention which – nevertheless – is highly specific to the image selection. This is possible because pictures of natural or artificial scenes in general contain a lot of redundant information. In particular, the analysis of natural scenes suggests that spatially related pixels are strongly correlated [Simoncelli and Olshausen, 2001, p. 1199], that is, knowing one pixel, its neighbours can be predicted with a high probability. This characteristic stems from the quite specific probability distribution which describes natural scenes (ibd.). Consider, for example, generating images by randomly assigning brightness values to their pixels. The probability of generating an image that could be classified as a natural or artificial scene is exceedingly low in this experiment. If the view is further restricted to a certain kind of scenery it should be obvious that the images of this scenery can only reside in a tiny subspace of the available pixel-brightness space.

Knowing that there is much less information in an image than if all pixels were uncorrelated, it should be possible to build visual features that identify an image in a certain scenery with only a few numbers. These features could then be used to represent the original image in a more condensed, prepared form. More interestingly, it should be possible to create these features autonomously since they depend only on the probability distribution of the images. Yet, we still have to define what properties the features should have.

To define the desired properties of the features, consider an agent learning a task using only visual information.

- As the agent acts and thus changes the environment, the pictures of this environment change. Thus, it is important to describe the state of those parts in the picture that change, instead of fully describing the picture, to learn about the meaning of the agent's actions.

- Moreover, as changes take place in the picture, the features should change smoothly. A smooth activation has two main advantages. First, it preserves a similarity relation between images. An analogous preservation can be found in tuning curves of some biological neurons [e. g. Pouget et al., 2000], where the activation is similar for related images. Second, the continuity of the activation facilitates the approximation of functions over the features when the functions vary with the same process, a property highly desirable if one wants to use neural networks (e. g. multi-layer perceptrons) for learning.

- As a simplification, we can assume that the important changes take place on a timescale similar to the one needed to solve the problem. This assumption provides us with a timescale on which we can look for changes and optimize the smoothness.
- Finally, it would be best if different features concentrated on different aspects of the image, for example signify the state of independent changes. If there actually are independent changes in the picture, then their combined state cannot be captured by one feature alone.

1.2. Aim of this Thesis

This thesis reports on an experiment testing the hypothesis that first, the network based on temporal stability and local memory described by Wyss et al. [2006] can generate the kind of features described previously from a stream of images, and second, that these features are indeed able to provide a neural network-based learning algorithm with input that eases learning. The testing ground will be a visual version of the pendulum swing-up task, which we will now briefly introduce.

The pendulum swing-up task (also known as “inverted pendulum”) is a classical reinforcement learning task (e. g. Santamaria et al. [1997], Doya et al. [2002]; an extensive discussion of the task’s complexity and possible solutions can be found in the work of Åström and Furuta [2000]). A pendulum is attached to a motor with the help of which the pendulum has to be swung (depending on motor strength, possibly with multiple swings) to an upright position. The task is widely used to test new learning algorithms because of its transferability to a physical system and its simplicity; particularly, the state of the pendulum can be completely described by two numbers, namely, its angle and its angular velocity. In addition, the number of actions is small, as the controller is usually given the choice of two actions, exerting a clockwise or counterclockwise force on the pendulum, respectively.

In this work, the dimensionality of the pendulum swing-up task is artificially increased by working on the visual representation of a pendulum. All possible images that can be generated from a pendulum angle $\theta \in [0, 2\pi[$ are part of a plain circle in $\{0, 1\}^n$, where n is the number of pixels. Both the current and last angle are transformed to an image, thus the possible states form a torus in $\{0, 1\}^{2n}$. Although each such combination of images can be described by just two numbers, the difficulty of learning the task increases tremendously with the number of pixels used[1], as mentioned above, since the correlations between pixels are at first not obvious to a machine learning algorithm. In a two-stage approach first, features will be generated autonomously that describe the state of the pendulum, and second, only those features will be used to learn to swing the pendulum to an upright position.

[1]Of course, if the resolution is too low, too much information about the state is lost and learning also becomes difficult.

2. Methods

Two techniques are coupled in this work, namely, unsupervised feature generation and reinforcement learning. In the following two sections, we will provide the knowledge about these techniques crucial for the understanding of the experiments.

2.1. An Unsupervised Network based on an Objective Function

Wyss et al. [2006] propose a layered network architecture optimized using an objective function comprising temporal stability and local memory. They suggest that optimization based on this objective function could be a general principle of information processing in the brain and could give rise to high level cortical functions. To demonstrate the relevance of their work, the authors show that it is possible to generate so-called place cells[2] using the proposed network architecture.

In this thesis we will make use of a simplified version of the network architecture and the learning principle proposed by Wyss et al. [2006], using it to extract features from images. Hence, we will shortly summarize the essential ideas.

The input, usually an image taken from a stream of images, is fed forward as $\overrightarrow{I}^{\,l}$ consecutively through layers $l = 1, 2 \ldots$ containing a decreasing number of units. Each unit i has two subunits with a local view onto the lower layer, limited by a circular receptive field and represented by the weight matrices $\overrightarrow{W}_1^{l,i}$ and $\overrightarrow{W}_2^{l,i}$. The activation of unit i in layer l for timestep t is then determined by

$$A_l^i(t) = \sqrt{\left(\overrightarrow{I^l}(t) \cdot \overrightarrow{W}_1^{l,i}\right)^2 + \left(\overrightarrow{I^l}(t) \cdot \overrightarrow{W}_2^{l,i}\right)^2}. \tag{1}$$

As the number of neurons decreases with every layer, the size of their receptive fields is increased such that the receptive fields cover the whole image.

For an incoming stream of images, the weights matrices $\overrightarrow{W}_1^{l,i}$ and $\overrightarrow{W}_2^{l,i}$ are continuously changed to optimize the unit's activation over time according to two criteria.

1. The activation of one neuron should change smoothly over time. This corresponds to maximizing the stability

   $$\mathcal{S} = -\sum_i \frac{\left\langle \left(A_l^i(t) - A_l^i(t-\tau_l)\right)^2 \right\rangle_t}{\mathrm{var}_t(A_l^i)}, \tag{2}$$

   where $\langle \cdot \rangle_t$ denotes temporal averaging. To avoid the trivial solution of an unchanging activation over time, the activity is divided by its variance. The layer parameter τ_l is increased with l in the original work to facilitate a change in time scale of higher layers by forcing them to abstract from short-term fluctuations of the lower layer.

[2]Place cells are a type of cell found for instance in the hippocampus of mice. Cells of this type are active when the animal is in a certain location regardless of the direction in which it is looking [i. e. McNaughton et al., 1983].

2. Neurons with similar receptive fields should try to capture different aspects of the stimulus. This condition is enforced by maximizing the decorrelation

$$\mathcal{D} = -\sum_{i \neq j} \left(\frac{\mathrm{cov}_t(A_t^i, A_t^j)}{\sqrt{\mathrm{var}_t(A_t^i)\mathrm{var}_t(A_t^j)}} \right)^2 , \tag{3}$$

that is, the squared temporal correlation between units, with $\mathrm{cov}_\mathrm{t}(\,\cdot\,)$ denoting covariance over time. As in the original work, only units sharing common feed-forward input are decorrelated by this term.

Putting both aspects together, the overall aim of the unsupervised learning is to maximize the objective function

$$\Psi = \mathcal{S} + \beta\mathcal{D}, \tag{4}$$

a weighted combination of stability and decorrelation of the activations within each layer. The units in the network then constitute visual features, filters that capture different aspects of the images and vary smoothly over time.

Before being passed to the next layer $l+1$, the activation of all units in layer l is transformed to

$$\overrightarrow{O}_l(t) = \frac{1}{\tau_l}\overrightarrow{A}'_l + \left(1 - \frac{1}{\tau_l}\overrightarrow{O}(\tau_l - 1)\right), \tag{5}$$

with

$$A'(t) = \frac{A(t) - \langle A \rangle_t}{\mathrm{var}_t(A)}. \tag{6}$$

That is, first the changes in the unit activation are weighted by the parameter τ_l. Note that again, setting $\tau_l \gg 1$ is intended to change the time scale for the next level. Second, the output is mean corrected and normalized to unit variance. Both (and only these) calculations incorporate a time-dependency (the "local memory") into the activation of neurons in the network, while the aforementioned temporal averages were only part of the learning rule.

2.2. Reinforcement Learning

The features generated by the unsupervised network described in the previous section can serve as the input of a controller. With the help of a signal indicating success or failure, this controller can then be taught to accomplish a task. The techniques of teaching the controller are subsumed by the term "reinforcement learning", which will be briefly introduced in the following.

A Markovian Decision Process (MDP) is a framework to model sequential actions of an agent in its environment. It is defined as a 4-tuple, including

- a set of states $s \in S$
- a set of actions $a \in A$,

- a function $p : S \times A \times S \to \mathbb{R}$ defining the probability of a transition from state s to state s' if action a is chosen, and
- a function $c : S \times A \to \mathbb{R}$ defining the immediate reinforcement signal that the agent receives when carrying out action a in state s.

MDPs feature the Markov property, that is, the probability of a transition to state s' is only determined by the current state s and the chosen action a and explicitly not by previous states and actions. This property is emphasized, since it motivates some changes to the unsupervised network. In this thesis, we allow S to be continuous and assume that A is finite. We further assume that p is unknown to the learning system.

The "solution" of an MDP is an optimal policy $\pi^* : S \to A$ mapping states to actions, that minimizes the costs defined by $c(\,\cdot\,)$ if the agent acts according to it. In the case of an unknown transition function p, the task of finding π^* can be achieved by Q-learning.

The Q-learning algorithm [see e. g. C., May 1992] estimates the expected costs for a state/action pair by sampling transitions from one state to another. The estimate is given by the Q-function $Q(s, a)$, which is usually updated according to the rule

$$Q_{k+1}(s,a) \coloneqq (1-\alpha)Q_k(s,a) + \alpha(c(s,a) + \min_b Q_k(s',b)) \tag{7}$$

for problems with a finite horizon, which we are concerned with in this thesis. The usual approach is to update the Q-function online, after each observed transition.

When the Q-function is approximated by multilayer perceptrons, the online updating approach performs badly. The reason for the bad performance is that multilayer perceptrons cannot be adjusted locally without changing their output for other areas of the state-action space in an unpredictable manner. Hence, updates after each transition have to be made very carefully (α is small in Equation (7)), which results in small learning rates and the need for many trials.

The Neural Fitted Q-Iteration (NFQ) algorithm [Riedmiller, 2005a] deals with this problem by remembering all observed transitions and reusing them for training. Consequently, updates can be committed "offline" and algorithms for batch training of the multilayer perceptron can be employed, resulting in a more stable learning process. The main idea of NFQ is to use the recorded samples to generate a training set, which is based on the current estimate of the Q-function Q_k, i. e. as if the recorded transitions were observed again. In particular, for each transition (s, a, s'), we set

$$\begin{aligned} \text{input} &= (s,a) \\ \text{target} &= c(s,a,s') + \min_b Q_k(s',b) \end{aligned}$$

and collect all pairs (input, target) in the training set. This training set can then be used to adjust the estimate of the Q-function Q_{k+1} with the help of e. g. Backpropagation or RPROP [Riedmiller and Braun, 1993]. Note that this procedure can be repeated without observing new transitions since the recorded transitions and the new estimate of the Q-function can be used to generate a fresh training set. We will make use of the NFQ algorithm to accomplish the reinforcement learning tasks in this paper.

3. An Architecture for Visual Task Learning

With the help of the discussed methods, it is now possible to describe the architecture with which we will tackle the visual task learning problem as a Markovian Decision Process.

- We suppose that we are given a system determined by a probability matrix $p(s, a, s')$. Here, the states s and s' contain system parameters which are not known but can be observed indirectly via a video stream.
- The state space S is formed by the features of the topmost layer of the unsupervised network and their history of activation of length h

  $$S = \left\{ s | s = \overrightarrow{A}_0, \overrightarrow{A}_1, \ldots, \overrightarrow{A}_h \right\}.$$

  Note that h needs to be adjusted to the observed system: Some parameters of the system can not be observed in one image, but setting $h = 1$ enables the learner to calculate differences between the last and the current state, $h = 2$ allows to determine the rate of this change, and so on. Additionally, higher values of h can be useful if the rate of change is small compared to level of noise [Hernandez-Gardiol and Mahadevan, 2000].
- Further, suppose that we can perform actions $a \in A$ that modify the system. As actions are executed, we can observe the new state of the system s' indirectly through the images. Consequently, we can apply the technique of Wyss et al. [2006] to find visual features that vary smoothly with the image stream as it changes.
- Finally, we assume that there is an immediate reward function $c(\,\cdot\,)$ based on the image or the system state itself.

For the features to constitute the state of an MDP, the unsupervised network as introduced in Section 2.1 needs some alterations. In the original work, the local memory constant τ_l was chosen to increase with the layer number. Since $\tau_l > 1$ introduces an infinitely large time dependency unsuited for MDPs in Equation (5), in this work we set $\tau_l = 1$ for all l. This effectively eliminates the second term in Equation (5). More severely, it deprives the network of the ability to ignore high-level redundancies in the data, but these are simply not present in the pendulum swing-up task[3]. For problems where abstraction is desired $\tau > 1$ in combination with a moving window of fixed length could be used instead of the infinite window. Finally, the time dependence introduced by the functions $\langle \cdot \rangle_t$ and $\mathrm{var}_t(\,\cdot\,)$ in Equation (6) was kept for the training of the unsupervised network. During its evaluation, however, their values had to be fixed to the value at the end of training.

[3]This point was labelled "severe" as for the original paper this would mean that no place cells (see Footnote 2 on page 5) could be found, due to the fact that the abstraction from the current view to the current position was accomplished with the help of an increasing time constant in the neurons activation resulting in different time scales across layers.

We modeled the visual task learning problem as an MDP in order to be able to perform reinforcement learning on it, which in turn has numerous variants that have to be decided on.

- We used a variant of Q-learning, as Q-learning can be used without a model of the system, which we assumed to be unknown since we want to restrict ourselves to the camera as a source of information.

- Although the dimensionality of the visual task can be reduced significantly by the feature extraction process, it remains too high to represent the Q-function in a look-up table with reasonable space requirements. For instance, later we will use nine features and a history of length two. A look-up table with a resolution of just 10 in each dimension would therefore have an excessive number of 10^{18}, entries. Using table-based function approximators such as CMACs [Albus, 1975] the space requirements can be reduced, but the general space weakness of tables, their exponential growth with the input dimensionality, is not eliminated. Therefore, we decided that the Q-function should be represented in a multi-layer perceptron, which does not suffer from this limitation.

- Training Q-functions represented by multilayer perceptrons is difficult since newly acquired information cannot easily be incorporated locally into the Q-function without changing the function as whole. The NFQ algorithm (Section 2.2) is able to reuse previously acquired information to overcome this drawback and was therefore chosen to realize the task of learning.

4. Results

To gain experience with the setup proposed in Section 3, we decided on a simple testing ground, a simulated inverted pendulum. As described above, we created a virtual image stream by projecting the pendulum onto a virtual camera image as it moved and passed the images to a network of an architecture adopted from Wyss et al. [2006] which was designed to optimize the activation of its neurons for stability and decorrelation. Since we chose the layers of the network to decrease in size, at the same time the original image was transformed to a much smaller representation. Based on this reduced representation and an external success signal, we derived a controller performing the pendulum swing-up task and optimized it using the Neural Fitted Q-Iteration reinforcement learning algorithm [Riedmiller, 2005a]. The whole setup is visualized in Figure 1.

The following experiments were performed using an augmented version of *wSim*, a network simulator programmed by Reto Wyss. We enhanced the functionality of the simulation environment with a configurable, extensible and thoroughly tested library which is able to carry out several variations of Q-learning (online and offline learning, Advantage Updating, Neural Fitted Q-Iteration, additional parameters) on several function approximators (Tables, CMACs and Multi Layer Perceptrons).

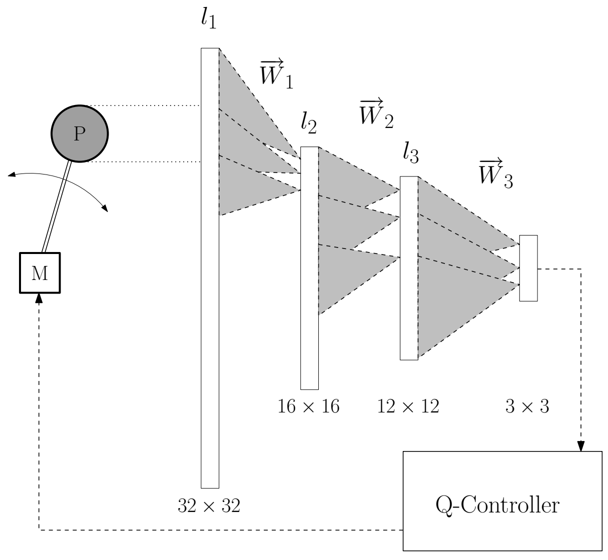


Figure 1: The simulated experimental setup. The motor M controls the movements of the pendulum P. The projection of the pendulum ("camera image") is fed to a feed forward network of three layers decreasing in size. The output of the last layer is used as a state representation for the Q-controller. Closing the loop, the Q-controller sends motor commands to the motor.

4.1. Description of the Stimulus

The experiments were performed on a simulated inverted pendulum represented by a mass point. For comparability, the physics parameters of the pendulum simulation (i. e. mass of pendulum, length of pole, integration time) were adjusted in such a way that the task of swinging the pendulum to an upright position was approximately as hard as in the real-world task described by Riedmiller [2005b]: An optimal controller (derived by executing reinforcement learning directly on the angle and velocity) needed about two swings – corresponding to 20 simulation steps – to bring the pendulum from the initial position into the target area. To determine the physical parameters mentioned above, the initial position of the pendulum was accordingly set to resting (pointing downwards) while the target area was defined as the top position ± 0.3rad.

The generated stimulus consisted of a parallel projection of a white disc at the pendulum's position onto a virtual 32×32 pixel camera image. The diameter of the disc was chosen to be 30% of the virtual camera image width. Pixels representing the pendulum were assigned a fixed luminance at 100%. The background pixels were initialized with random luminance (10%, $\sigma = 30\%$) and unchanged during the experiments.

Before actually being presented to the main network, the stimulus was passed through an edge detection filter. Note that this was just kept for comparability. Actually it made the task of analyzing the image statistics more difficult for the pendulum example since the stimulus was transformed from a "blob" to a ring.

4.2. Feature Generation in the Unsupervised Network

In a preparatory phase, the unsupervised layer was trained. We initialized the weights by sampling from a normal distribution with $\mu = 0$ and $\sigma^2 = 1.25$. Then, the pendulum

	Smoothness	
Layer No.	untrained	trained
1	-2.592	-2.493
2	-5.988	-2.498
3	-26.572	-2.025

Table 1: This table displays the average stability of neurons in all layers obtained with Equation (2) using a non-weighted average over one pendulum rotation. The values increase with training. This effect is stronger in higher layers.

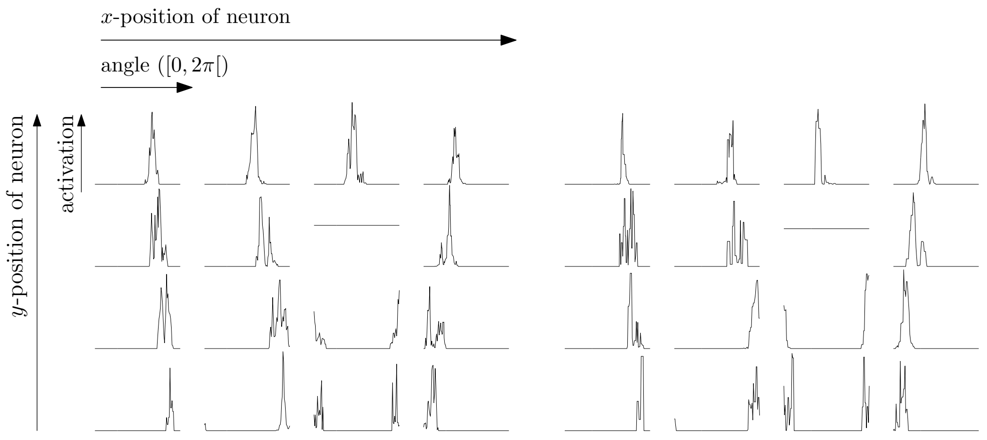
was presented to the network in a monotonous circular ($1°$/timestep) motion. In this phase, the neurons of the unsupervised network adjusted their weights according to the objective function, i. e. in such a way that their activation changed slowly while the pendulum was moving and close-by neurons were decorrelated as required by Equations (2) and (3). Temporal averaging was performed with a time constant of $\tau = 1000$. The weight of the decorrelation β in the objective function (Equation (4)) was fixed to 0.1 across layers. It could not – although suggested in Wyss et al. [2006] – be further increased with the decreasing number of neurons per layer due to a property of the simplicity of the pendulum swing-up task: There is almost no uncorrelated activity in the image because the stimulus is just one moving "blob" and all activity is necessarily correlated to its movements. Nevertheless, setting $\beta > 0$ avoids the possible outcome that all neurons with overlapping receptive fields acquire the same or similar tuning curves. The training process was stopped after 500000 time steps, when the value of the objective function had converged (Figure 3).

The initial (pre-training) and the resulting (post-training) tuning curve of some neurons from the three layers are depicted in Figure 2. From layer one to three, neurons were active for a growing interval of angles. Specifically, they could only change their activation when the edges of the pendulum fell into their respective receptive field (resulting in at least one neuron not changing its activation at all). It was only within this interval that optimization with respect to the objective function could take place. Comparing initial and resulting activations in Figure 2, we see that for the small receptive fields in layer 1 this optimization effect is hardly noticeable (a-b), while activations of neurons in layers 2 and 3 (c-d and e-f, respectively) are visibly more stable after training. These relationships are quantified in Table 1, which displays the average stability for every layer before and after training. Here, stability was calculated with Equation (2) using a non-weighted average over one pendulum rotation ($1°$/timestep).

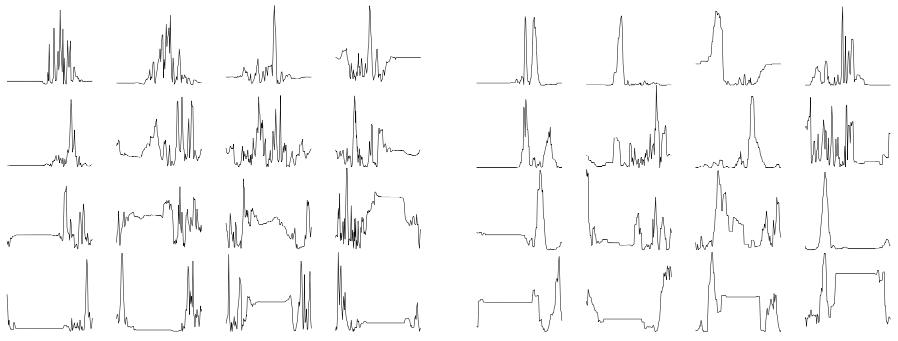
4.3. Reinforcement Learning Using Generated Features

For the following reinforcement learning experiments, learning was turned off in the unsupervised network, thus fixing the weights to their current value. Additionally, the mean and variance over time of the neuron's activation (Equation (6)) were fixed to their post-training values, so that the activation no longer depended on the previous activations. This ensured the Markov property (Section 2.2) of the problem.

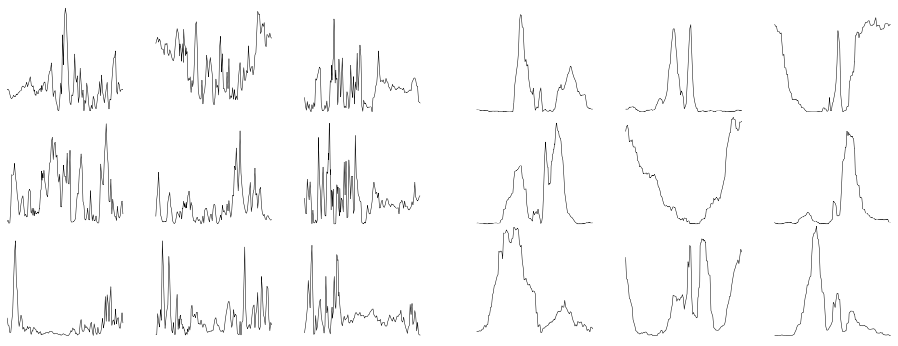
The trained unsupervised network was tested for fitness as a state representation in



(a) Untrained neurons in layer 1 (b) Trained neurons in layer 1

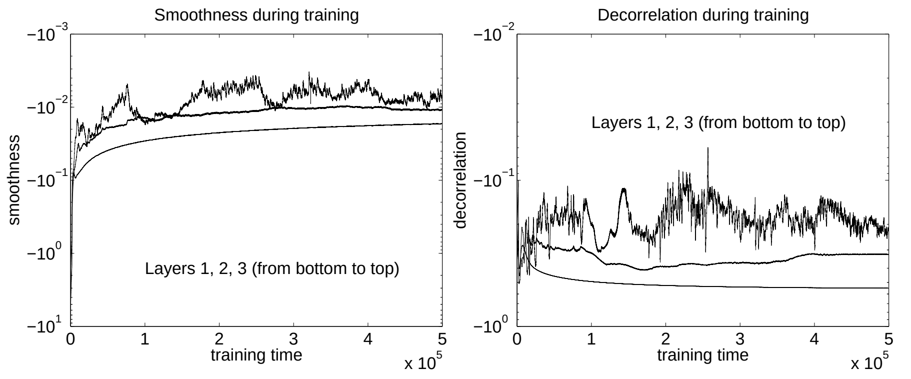


(c) Untrained neurons in layer 2 (d) Trained neurons in layer 2



(e) Untrained neurons in layer 3 (f) Trained neurons in layer 3

Figure 2: The figure shows the tuning curve of neurons in the unsupervised network as a mapping from the angle of the pendulum (x-axis) to the neuron's activation (y-axis). Only a subset of the neurons is shown for layer 1 and 2. The position of the activation plots within a block correspond to the position of the neuron with respect to the camera image. Their increasing activation across the layers reflects the enlarging receptive field.



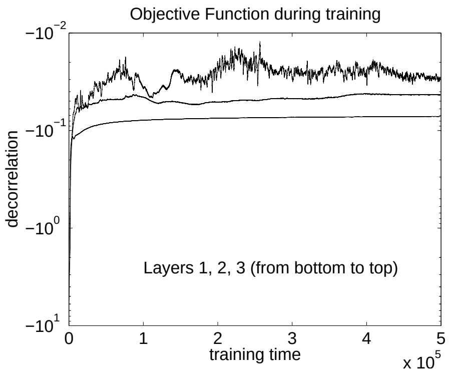


Figure 3: The top figures show the development of stability $\mathcal{S}$ and decorrelation $\mathcal{D}$ over time. Below, their weighted sum $\Psi = \mathcal{S} + 0.1 \cdot \mathcal{D}$, as it was maximized in the experiment, is displayed.

a neural net-based reinforcement learning task. For this purpose, a Q-Controller was added to the otherwise strictly feed-forward network architecture. According to the output of the unsupervised network, it had to decide whether to exert a clockwise or counterclockwise force onto the pendulum by observing the effects of its motor commands for sampled states and generalizing for the others. For training, the pendulum was always initialized to a position pointing downwards with zero velocity. The current and the last activation of the top layer ($3 \times 3 = 9$ features) were passed to the controller, implicitly enabling it to derive the position and the velocity of the pendulum. Therefore, the pendulum state could be described in $2 \times 9 = 18$ dimensions. Since the activation varied strongly in amplitude across features, each feature was normalized such that its value ranged between -1 and 1.

To show that no special requirements are necessary for the method proposed in this paper, the Q-function used for the controller was represented in a similar network architecture as described by Riedmiller [2005b]: The state description was fed into a multilayer perceptron with two hidden layers consisting of five units each, with the hyperbolic tangent as their activation function. Unlike in the aforementioned work, for each action one such network was used to represent the action's Q-values and the single output unit was activated linearly.

While the controller greedily exploited the Q-function without exploration, a fixed cost of 0.01 for every step was given and state transitions were recorded for NFQ training. When the controller succeeded in its task, the cost of the step leading into the goal region was reduced to zero. At the same time, the Q-value of the state that was now discovered to be part of the goal region of the state space was set to zero and excluded from NFQ updating to stabilize it there. The training was stopped after 100 transitions, or whenever the controller succeeded. Before running the next trajectory, two iterations of the NFQ algorithm were applied. In each iteration the Q-function was adjusted to the newly determined Q-values with 1000 epochs of RPROP-training [Riedmiller and Braun, 1993]. The performance of this controller is visualized as "trained" in Figure 4.

To be able to evaluate the results, the same task was also learned with three different state spaces, while all parameters not mentioned were kept as above:

1. Untrained features ("untrained" in Figure 4). After randomly assigning weights to the unsupervised network, the variance and mean activation as needed for Equation (6) were determined by rotating the pendulum until they converged and then fixed. As before, the features of two successive pictures formed the $2 \times 9 = 18$ dimensional state.

2. Angle and velocity of the pendulum ("ang-vel" in Figure 4). The state variable obviously has two dimensions. To some extend, the resulting controller constitutes an upper bound for the others w.r.t. the performance.

3. Raw pixels ("pixels" in Figure 4). The pixels of the virtual camera (current and last image) make up the state, which has a dimensionality of $2 \times 32 \times 32 = 512$.

The graphs in Figure 4 are the result of averaging over 20 repeated experiments with different initializations of the neural net representing the Q-function. The error bars display the standard deviation over these trials. The results can be summarized as follows:

- Reinforcement learning on the features ("trained") yielded a policy which was worse than learning on directly acquired pendulum parameters ("ang-vel") in terms of swing-up time (on average 8.7 time steps more during the last 20 trajectories). Nevertheless, it converged after about 45 trajectories, similarly fast compared to learning on angle and velocity. As can be seen in Figure 5, a sub-optimal policy requiring 30 steps to swing up the pendulum was reached within 1100 interaction time steps for both methods.

- The performance of the feature-based controller varied more across trials than the performance of the controller based on angle and velocity ($\bar{\sigma}_{trained} = 10.5$, $\bar{\sigma}_{ang-vel} = 4.1$ time steps over the last 20 trajectories, respectively).

- The untrained unsupervised network did not seem to support learning at all.
- The raw pixel input supported learning only to a limited degree: The learning process was quite unstable ($\bar{\sigma}_{pixels} = 22.9$, during the last 20 trajectories) and converged slowly (the point of convergence was not determined due to excessively long training times).

Note that learning on pixels took about 40 times as long as learning on the trained features (30.9 hours and 0.8 hours, respectively, on a 2.6 GHz Dual Core AMD Opteron). For comparison, training on angle and velocity took 0.3 hours on the same computer. This difference is rooted almost entirely in the RPROP training time. However, worse controllers produce longer trajectories, resulting in more training data and thus longer computation time.

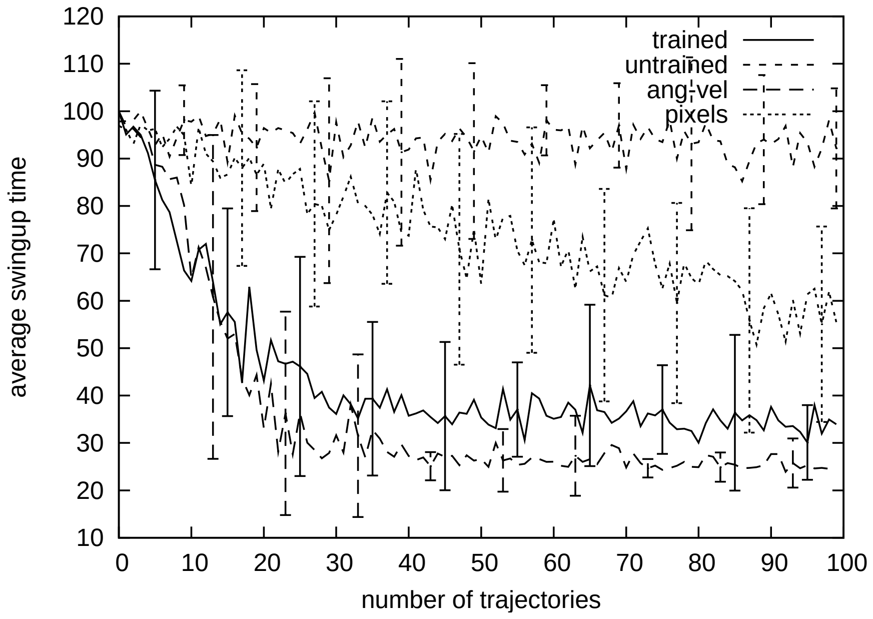


Figure 4: The figure shows the performance of learners based on continuous (“ang-vel”) angle and velocity, the trained/untrained unsupervised layer and the image pixels themselves. Depicted is the number of steps needed to swing up the pendulum depending on the number of training trajectories seen. Each data point is the result of an average over 20 trials. Error bars show the standard deviation over the same 20 trials.

4.4. The Impact of Stability on Performance

The whole experiment described in the previous sections (unsupervised feature generation and 20 trials of reinforcement learning) was repeated an additional eight times.

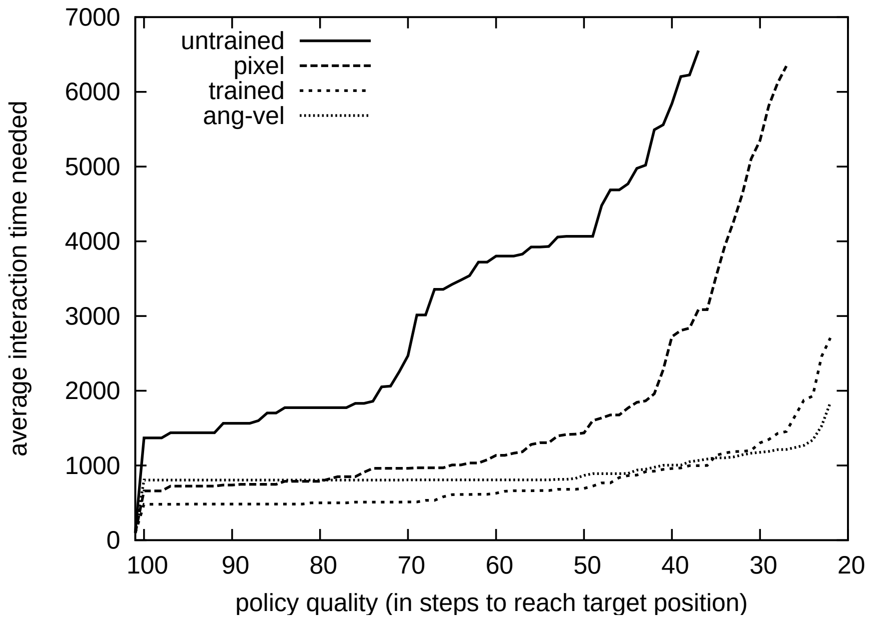


Figure 5: The figure shows the average interaction time needed to reach a policy of a certain quality once. If the quality was only reached in less than 50% of the trials, the line was discontinued. The lines show the behavior of controllers learnt on trained/untrained unsupervised neurons ("trained", "untrained"), the pixels of the image ("pixel") and on angle and velocity ("ang-vel").

Again, the performance was averaged, this time over the averages of the single experiments, and plotted (Figure 6). We note that the performance which we recorded for our first feature set seems to be no exception, since the newly generated features yield equally good results. However, there are differences in the quality of the reinforcement learning controller.

We analyzed whether the performance of the reinforcement learning controller varied with the stability. This was done in the following steps:

1. We calculated the average stability of all generated feature sets with Equation (2) using a non-weighted average over one pendulum rotation.
2. We determined the mean performance of the respective reinforcement learning controller. This was defined as $\langle p_t \rangle$, where p_t is the average performance of the controller in 20 reinforcement learning trials at the time t, with $t \in \{1 \dots 100\}$.
3. Similarly, we calculated the average variance over 20 trials at time t, as in $\langle v_t \rangle$.
4. Finally, we plotted the mean and its variance as a function of the stability and fit a line to this data. This is shown in Figure 7.

Both the mean performance and its variance seem to depend on the stability. The swing-up time decreases when features have stable tuning curves, as does the variance.

Please note that there are only seven, not eight, data points in Figure 7. We eliminated one outlier after examining it more closely: The features displayed in Figure 9h (Appendix) have low stability, but good performance. Apparently this results from a inhomogenity of the features, as some are much more stable than others. Consequently, the variance of the stability in this set is much higher than in the others. This shows on the one hand that (as mentioned) there is redundant information in the features. On the other hand, it shows that the reinforcement learning was robust enough to ignore the less stable features. At this point, we are not interested in either of these questions, so it is safe to ignore this outlier.

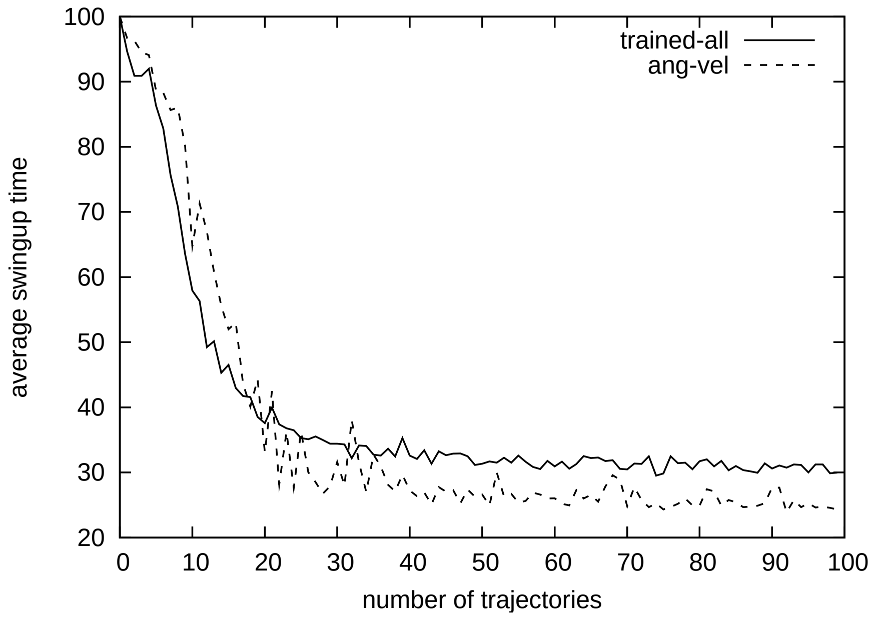


Figure 6: This figure shows the average performance of eight completely relearned controllers, including feature generation and reinforcement learning ("trained-all"). Across the controllers the performance is stable, which demonstrates the robustness of the learning algorithm. For comparison, the performance of the optimal controller is plotted again ("ang-vel").

4.5. Effect of the Discretized Image

The resolution of the unsupervised network was limited by the resolution of the virtual camera. To quantify this limitation, the pendulum was moved in a full circle once in tiny (0.01°) steps. The visual image did not change with every such movement, as the

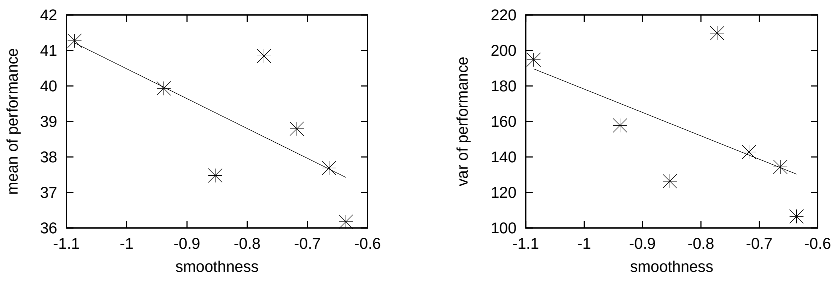


Figure 7: This figure shows the mean performances (left) and the variance of the performance (right) of the reinforcement learning controllers as a function of feature stability. The lines were fitted using the least-squares algorithm applied to the parameters a, b of the function $f(x) = ax + b$. A higher feature stability seems to result in both better performance (i. e. lower swing-up time) and lower variance.

resolution was too coarse. For the pendulum used in this experiment, with a diameter of 30% and a length of 40% of the image width, and a camera resolution of 32×32, the number of different states was 1240. One such state was determined to correspond to an angle of $0.3°$ with a standard deviation of $0.3°$.

To test the effect of discrete angle perception, another control was performed, training on angle and velocity discretized to 1240 equal sized buckets. Contrary to the assumption that this would worsen the policy, the performance after convergence and the convergence speed are similar or better (Figure 8). This indicates that the level of performance of the visually trained controller is not a result of the discretization. Rather, the reason for the performance loss lies either in the information lost in the unsupervised network, the still increased dimensionality or the more complicated structure of the state data.

5. Discussion

The feature extraction process is very general and can easily be applied to other problems. Since the features are optimized for stability and decorrelation based on the stream of images, no task knowledge is required. In particular, the exact dimensionality of the original problem (except for the size of the history required to solve it) need not be known. Indeed, a higher task dimensionality than the one of the pendulum swing-up problem would definitely increase the understanding of the discussed method, since then there might actually be independent movements in the picture that could be sensibly decorrelated by the feature extraction process.

In this work, the feature extraction step was performed separately from the learning of the task. The reason for this decision is rooted primarily in methodological constraints, but the two-stage-approach has some notable parallels in the literature as well. In the following, we will discuss both points.

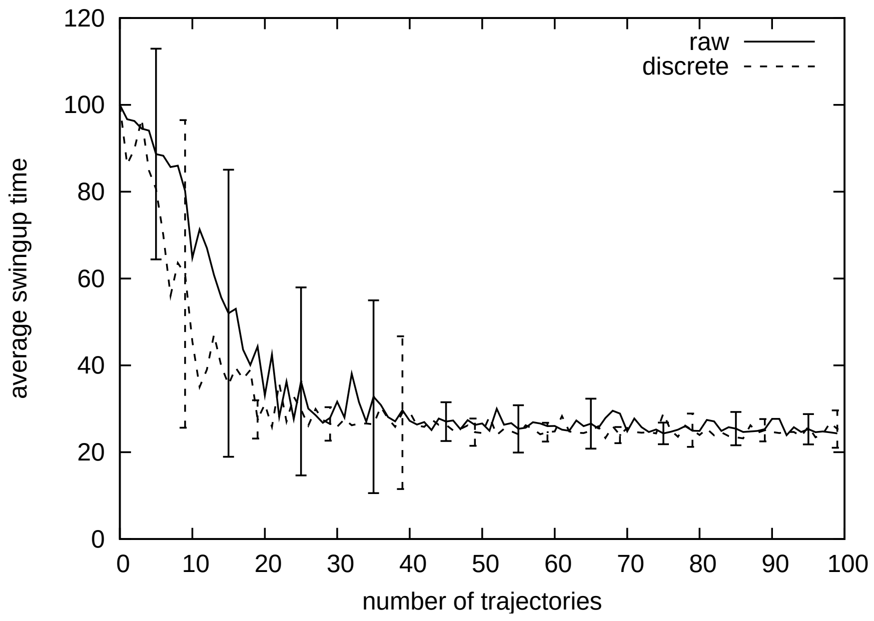


Figure 8: This figure shows the performance of two controllers, one learning on continuous angle and velocity ("raw"), one learning on angle and velocity discretized to the same extent as done by the resolution of the camera image ("discrete"). Each data point in the graph represents the average of 20 trials, the error bars show the standard deviation over these trials.

1. During the feature extraction process the semantics of the features change continuously. This is not compatible with the Neural Fitted Q-Iteration (NFQ) algorithm used for the reinforcement learning task, as it reuses *all* experiences. Thus, even if we suppose that newly explored subspaces of feature space do not overlap with previously explored ones, that is, there are no inconsistencies, NFQ would still try to accommodate its Q-function to all previous experiences. Clearly, the chosen neural net function approximator is inadequate for this purpose. Nevertheless, it seems desirable to run both the feature extraction and the reinforcement learning at the same time, such that as the reinforcement learning process learns to explore new areas in stimulus space features for these areas can be created. For this to succeed, some kind of forgetting needs to be implemented. The obvious idea, to implement this forgetting depending on the stability of the feature tuning curves f (forget fast in case of low stability and forget slowly in case of high stability to enable fine tuning), is deceptive, however. In particular, to calculate this tuning curve stability function, that is, the change in the mapping $\theta \times A_l^i \to f$, one would need knowledge about the system parameters, θ, during training. Consequently, one major property of the proposed method,

that no knowledge about θ is needed, would be lost.

2. It seems possible to provide a biological reason for a lengthy feature extraction phase which is not related to task learning directly. Studies of sleeping patients suggest that the activity of visual areas is dissociated from the actual physical input at the time of REM sleep [Braun et al., 1998]. In principle, this leaves room for some kind of repetitive replay of seen inputs. Furthermore, a good night's sleep seems to tremendously help the improvement of motor skills [Walker, 2002] and visual discrimination skills [Gais et al., 2000]. Both skills together are the key to successful learning in the conducted experiment.

For this work, the unsupervised network was deprived of one of its main properties, its ability to abstract from low level states. This was possible because for the pendulum swing-up task abstraction was not necessary. Nevertheless, if implemented, the layers with higher abstraction could be used in complex reinforcement learning setups to decide on macro actions which in turn are guided by layers with lower abstraction [analogous to Sutton et al., 1999].

To learn the task of swinging up the pendulum, no special demands were made on reinforcement learning. In particular, the same learning parameters and network structures – aside from state dimensionality – could be used for learning on the extracted features and for learning directly on the state of the pendulum. The only difference to classical reinforcement learning is that the semantics of the generated features are not known in advance. Consequently, the cost function, or, more specifically, the decision of whether the task succeeded, cannot be determined based on the generated features, it has to be provided by an external observer who has direct access to the state of the system (e. g. a human), a handcrafted visual feature of the camera image, or a different sensor (e. g. a switch or a collision sensor). However, if one confines oneself to the creation of a feature deciding whether the goal state has been reached, the effort should still be greatly reduced compared to the one needed for creating features for the whole task.

6. Conclusion

This thesis describes a method to generate visual features from a time series of images which are fit for neural net-based reinforcement learning. To demonstrate its effectiveness, the method was applied in simulation to a visual version of the pendulum swing-up task.

We showed that learning on the trained features greatly improves performance over learning on the pictures themselves or untrained features. Thus, our method may help in situations where no other sensor data but visual information is available. Since the features are created in an unsupervised manner, they can also help if handcrafting of features is too difficult or not an option.

We further analyzed the performance of a neural-net based reinforcement learning algorithm, which had to learn given only the features created beforehand. The key property of our features, the stability, correlated with its performance, which supports our hypothesis that their output is particularly suitable for such algorithms.

Although the results are encouraging, further research needs to be done, especially with real-world data and more complex tasks.

References

J. Albus. Data Storage in the Cerebellar Model Articulation Controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control*, 93:228–233, 1975.

K. Åström and K. Furuta. Swinging up a Pendulum by Energy Control. *Automatica*, 36(2):287–295, 2000.

A. Braun, T. Balkin, N. Wesensten, F. Gwadry, R. Carson, M. Varga, P. Baldwin, G. Belenky, and P. Herscovitch. Dissociated Pattern of Activity in Visual Cortices and their Projections During Human Rapid Eye Movement Sleep. *Science*, 279 (5347):91–95, January 1998.

Watkins C. Technical Note: Q-Learning. *Machine Learning*, 08:279–292(14), May 1992.

K. Doya, K. Samejima, K. Katagiri, and M. Kawato. Multiple Model-Based Reinforcement Learning. *Neural Computation*, 14(6):1347–1369, 2002.

S. Gais, W. Plihal, U. Wagner, and J. Born. Early Sleep Triggers Memory for Early Visual Discrimination Skills. *Nat Neurosci*, 3(12):1335–1339, December 2000.

N. Hernandez-Gardiol and S. Mahadevan. Hierarchical Memory-Based Reinforcement Learning. *NIPS*, pages 1047–1053, 2000.

B. McNaughton, C. Barnes, and J. O'Keefe. The Contributions of Position, Direction, and Velocity to Single Unit Activity in the Hippocampus of Freely-Moving Rats. *Experimental Brain Research*, 52(1):41–49, 1983.

A. Pouget, P. Dayan, and R. Zemel. Information Processing with Population Codes. *Nature Reviews Neuroscience*, 1(2):125–32, 2000.

M. Riedmiller. Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. In João Gama, Rui Camacho, Pavel Brazdil, Alípio Jorge, and Luís Torgo, editors, *ECML*, volume 3720 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2005a.

M. Riedmiller. Neural Reinforcement Learning to Swing-Up and Balance a Real Pole. *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, 4:3191–3196 Vol. 4, 2005b.

M. Riedmiller and H. 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.

J. Santamaria, R. Sutton, and A. Ram. Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces. *Adaptive Behavior*, 6(2): 163–217, 1997.

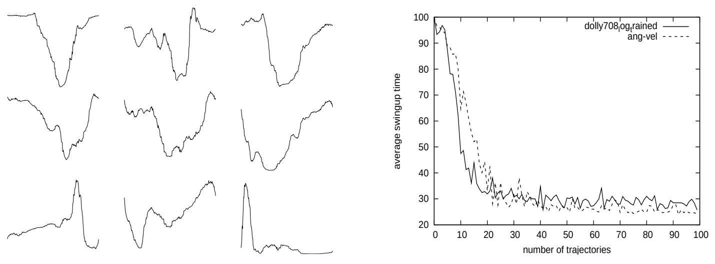
E. Simoncelli and B. Olshausen. Natural Image Statistics and Neural Representation. *Annu Rev Neurosci*, 24:1193–1216, 2001.

R. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112 (1-2):181–211, 1999.

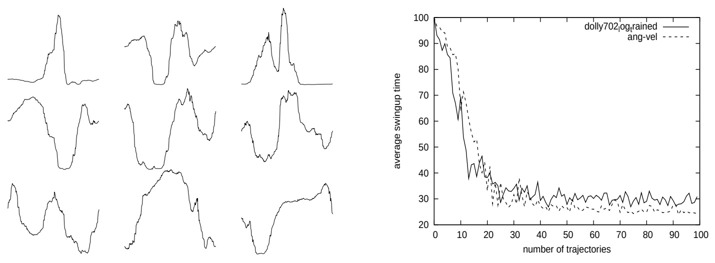
M. Walker. Practice with Sleep Makes Perfect - Sleep-Dependent Motor Skill Learning. pages 205–211, July 2002.

R. Wyss, P. König, and F. Verschure. A Model of the Ventral Visual System Based on Temporal Stability and Local Memory. *PLoS Biology*, 4(5):e120, May 2006.

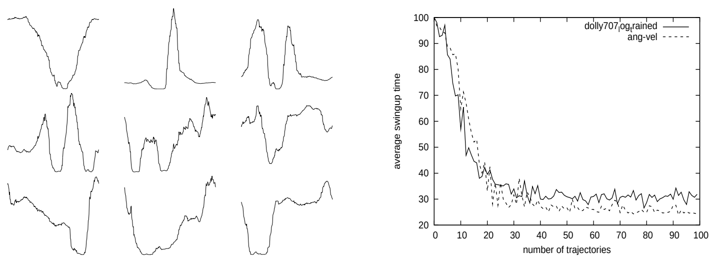
A. Appendix



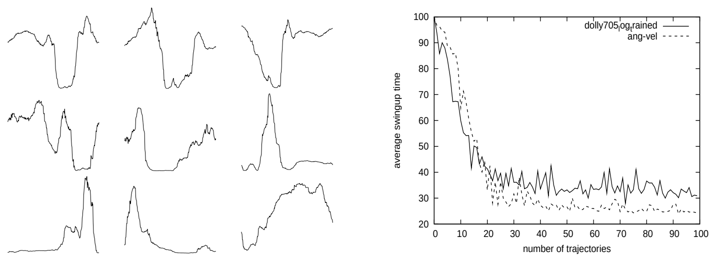
(a) smoothness = 0.63633



(b) smoothness = 0.66407

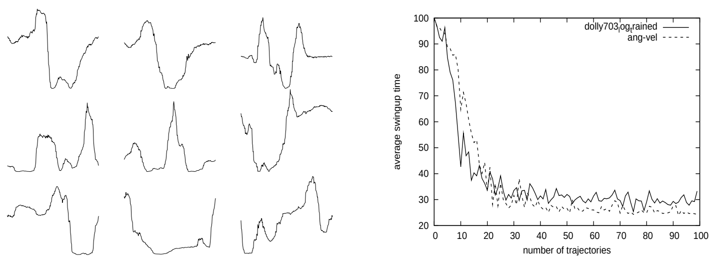


(c) smoothness = 0.71763

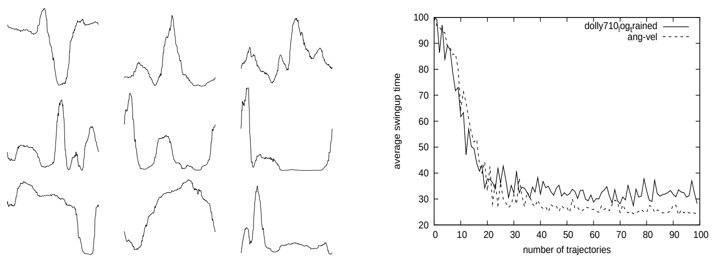


(d) smoothness = 0.77225

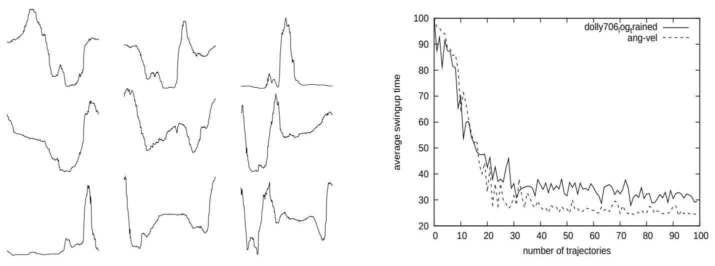
Figure 9: This figure shows the performance of features generated in separate trials. The subplots are sorted according to their stability. For comparison, the performance on the controller based on angle and velocity is also plotted.



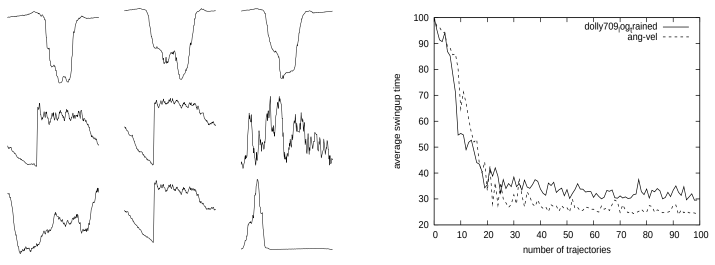
(e) smoothness = 0.85321



(f) smoothness = 0.93861



(g) smoothness = 1.08665



(h) smoothness = 2.33775

Figure 9: Continued

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Osnabrück, den 23. März 2007