Master Thesis

February 10, 2011

Avoiding failure states during Reinforcement Learning

Delft Biorobotics Laboratory, Department of 3ME,
Delft University of Technology

Author:
Merel van Diepen

Supervisors:
Prof. dr. ir. Pieter Jonker
Ir. Erik Schuitema
Dr. Wouter Caarls

TU Delft

project number 1134

Abstract

The Delft Biorobotics Laboratory develops bipedal humanoid robots. One of these robots, called LEO, is designed to learn to walk using reinforcement learning. During learning, LEO will make mistakes and fall. These mistakes can cause serious damage to the system but are an integral part of the learning process. A likely solution is punishing the robot more severely for falling. However, punishing the robot too rigorously can lead to a robot that is too cautious to make a step. In this research, three methods were tested that reduce the amount of falls during learning, without restricting the possible solutions or increasing the learning time.

We introduce Threshold Restricted Learning (TRL), a new method for action selection that, during exploration, makes the probability an action is chosen dependent on the expected reward for taking that action. Actions with expected rewards below the set threshold have a significant reduced probability of being chosen. The concept of TRL developed from the desire to optimize the use of a pre-learned solution. Hence, TRL was tested after pre-learning in simulation. The largest reduction in the amount of falls achieved by TRL in this research was 50% TRL did not increase the learning time. Without pre-learning the largest reduction was 7%.

Softmax action selection is a well known but underused selection method. Combined with pre-learning it was able to reduce the amount of falls during learning with approximately 80% and did not increase the learning time either. Without pre-learning, Softmax could still achieve a reduction of approximately 20%.

Sarsa$_{2Q}$, another new method, stores expected rewards at different levels op generalization. The level with little generalization is used to store the expected total positive reward, while the level with more generalization is used to store the expected total negative reward. Sarsa$_{2Q}$ learned to avoid the failure states for the inverted pendulum problem faster than using a single level of generalization. This method does not use pre-learning. The highest achieved reduction in the amount of falls was approximately 20%.

Sarsa$_{2Q}$ can have a broader use than just avoiding failure states. Rather than only using the coarse generalization for negative rewards, it can be used for all rewards that need less precision.

Acknowledgements

This thesis was an expedition into the jungle of reinforcement learning. A jungle with C++ swamps to get stuck in and with bugs that sting ruthlessly. And with a vastness that makes it impossible to explore in any scheduled amount of time. But the summits that were reached, sometimes by accident, provided an incredible view and a glimpse of the future. The lessons learned and the experience gathered were priceless. This report holds the, sometimes surprising, results of this journey.

I owe many thanks to my supervisors ir. Erik Schuitema and dr. Wouter Caarls because they showed me around the block, provided me with many ideas and valuable feedback and were my ultimate bug repellent. I would also like to thank Prof.dr.ir. Pieter Jonker for his efforts to broaden my field of vision. Special thanks goes to ir. Bart van Vliet for giggling when I did something stupid.

Finally, I would like to thank my lunch-friends for their company, my parents for their patience with my academic progress and Bart Bisschops for all the good times.

Delft, University of Technology
February 10, 2011

Merel van Diepen

Contents

Chapter 1

Introduction

Humanoid robotics is a developing field. Besides peoples' relentless fascination with the subject, the human body has some features worth copying. The gait of a human is still unmatched in its versatility and energy efficiency. The Delft Biorobotics Laboratory concentrates on bipedal humanoid robots. One of the robots, called LEO, is designed to learn to walk using reinforcement learning, [13].

Usually, when programming a walking robot, models of the system and the environment need to be available. The environment can not always be predicted and the system is generally too complex to be realistically modeled. By using reinforcement learning, one can avoid this problem because no models are needed. Reinforcement learning uses trial and error to learn which action to take in each situation. By rewarding good behavior and punishing bad behavior the robot learns to perform its task.

LEO can not fall sideways since it is fixed on a boom, see Figure 1.1. Therefore, the problem becomes two dimensional. LEO has actuators in the ankles, knees, the hips and the arm. During learning, LEO will make mistakes and fall. These mistakes can cause serious damage to the system but are an integral part of the learning process. A likely solution is punishing the robot more severely for falling. However, punishing the robot too rigorously can lead to a too cautious robot. In this case the robot will not be able to learn to walk. It is important to keep in mind that the goal of avoiding system damage is subordinate to the goal of walking; a damage avoiding strategy should not make walking unfeasible. This research focuses on avoiding falling itself, not on preventing damage to the robot.

It is important to notice the difference between fast learning and cautious learning. A method that learns fast (with respect to time) might have many very short episodes that ended in failure. Therefore, learning with a minimal amount of failure is not equivalent to learning in a minimal amount of time. But it is preferable when avoiding failure does not increase the learning time. Also, failure avoidance should not exclude possible solutions. E.g., requiring that the robot is stable at every time step will not allow for a dynamic gait.

This results in the following research question:

How can falling be avoided during learning on a walking robot, with minimal influence on the learning process?

In literature, several methods were found that might reduce the amount of falls during learning, namely Softmax Action Selection, Policy Gradient Reinforcement Learning, Model-based Reinforcement Learning and Hierarchical Reinforcement Learning. Next to these methods, two new

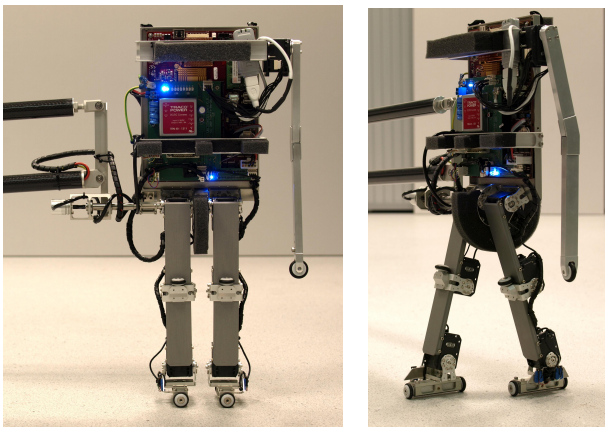


Figure 1.1: *Biped walker LEO [13]*

methods are presented, namely Threshold Restricted Learning and Sarsa$_{2Q}$. Policy Gradient Reinforcement Learning, Model-based Reinforcement Learning and Hierarchical Reinforcement Learning are not implemented in this research. They are discussed shortly here.
Policy Gradient Reinforcement Learning [14] represents the policy as a function of a set of parameters and updates the policy by a small step in the direction of higher total expected return. The design of the representation can be such that a group of useless policies is ruled out. This might help decreasing the amount of falls during learning. But with Policy Gradient Reinforcement Learning, the solution is steered by the possible forms the policy can take and next to that, prior knowledge of the system is needed. Because the gradient of the total expected return is calculated each step, it also has a high computational requirements. Because of these disadvantages this method is not implemented in this research.
Model-based Reinforcement Learning reuses previously observed experience between real steps. It simulates undergoing these experiences again, so the previously observed rewards can be propagated through the value function before new steps are taken. Therefore, Model-based Reinforcement Learning can learn with a smaller amount of experience, see e.g. Dyna learners in [15]. It might also have the potential of providing forecasts by simulating taking a sequence of actions. These forecasts can be consulted in order to determine if there are known failure states among the upcoming few states, provided the simulated sequence of actions is actually chosen. A state-action transition model of the system can be learned during operation using Model Learning techniques. It is doubtful that the amount of forecast calculations that needs to be done to reduce the amount of falls during learning can be performed in the small time span available during real time experiments.
Hierarchical Reinforcement Learning [16] can be used to introduce a supervisor to the system. This supervisor can switch from a module that governs the walking, to a module that limits the damage when failure is inevitable. This damage controlling module can be pre-programmed (non-learning) as is done by Fujiwara [11]. When the supervisor switches based on a programmed definition of inevitable failure, prior knowledge is needed and the solution is partly pre-determined. When the supervisor is also a Reinforcement Learning module, the learning time of the whole system is likely to increase and the advantage during learning may vanish. Hence, this method was not implemented.

In this thesis, the methods will not be tested on LEO or a simulation of LEO. Instead, a simulation of an inverted pendulum on a cart is used because learning on this simulation is fast

and the behavior of the methods is easier to predict and to analyze. This is due to the smaller state and action space. The simulation is described in detail in Chapter 3.

This thesis is organized as follows: In Chapter 2, the basics of Reinforcement Learning are explained. A description of the test simulation is given in Chapter 3. In Chapter 4, a new method, Threshold Restricted Learning, is explained and the test results are discussed. Chapter 5 presents the results for Softmax Action Selection. In Chapter 6, Sarsa$_{2Q}$, also a new method, is explained and the test results are discussed. Chapter 7 presents the conclusions of this report.

Chapter 2

Introduction to Reinforcement Learning

Reinforcement learning (RL) [1] is a sub-area of machine learning. Its approach differs from other methods in the sense that in RL, no example solutions to the problem need to be available, as is the case in supervised learning. RL algorithms learn from experience.
The RL problem consists of:

- a learner, also called an agent, that acts in the environment
- states, a state is a collection of all parameters that are relevant to describe the current state of the environment
- a reward function, a structure of positive and negative rewards linked to certain states or state-action pairs
- a policy, a table or function that defines the action to be taken in each state
- a value function, a function that holds the expected value of a state or state-action pair

The *learner* tries to maximize its total reward by finding the optimal *policy*. The policy describes the behavior of the agent, i.e., a definition of the actions it will take in each situation. The rewards are defined by a *reward function*. Through the reward function, the engineer can describe the goal of a task. Reaching the goal will result in a positive reward and taking a counterproductive action can be punished by a negative reward. Whereas the rewards indicate what the immediate value of an action is, the *value function* estimates the long term or total reward of following a policy.
In this chapter, the focus lies on Temporal-Difference Learning. Other forms of RL, like Dynamic Programming or Monte Carlo methods, need a model or can only update the expected values of states when the task is finished.

2.1 Basic Elements

In this subsection, the most important elements of an RL problem are presented.

Markov Decision Process Almost all RL problems are modeled as a Markov Decision Process (MDP). An MDP is a discrete process where at each time step an action is chosen that takes the process from one state to another. A process is an MDP when the state signal has the *Markov*

property, i.e., when in each state all the relevant information is available. This means that the effects of an action taken in a state depend only on that state and not on the prior history.
The MDP consists of a state-space $\mathcal{S}$, an action-space $\mathcal{A}$ and a state transition function $\mathcal{T}$ which describes the probability of reaching state $s' \in \mathcal{S}$ given an action $a \in \mathcal{A}$ and a current state s. The function that describes the state transitions is not necessarily deterministic. When using an MDP to model a RL problem, a reward function $\mathcal{R}$ is added to the representation. Such an MDP is represented by the tuple: $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$.

The Goal of RL The RL problem consists of finding a policy $\pi : s \rightarrow a$ that results in the maximum cumulative reward. The policy determines the action a that is executed given a state. The state-space $\mathcal{S}$ consists of all the possible states, while the action-space $\mathcal{A}$ consists of all possible actions in these states. Usually a policy is evaluated by either evaluating $V^{\pi}(s)$, the state value function, or by evaluating $Q^{\pi}(s, a)$, the state-action value function, and following the policy for all following actions. The state value function consists of the expected total reward from state s on. The state-action value function consists of the expected total reward from state s on, while taking action a and following the policy for all following actions. The difference between these methods is that the state value function $V^{\pi}(s)$ does not hold the transition function $\mathcal{T}$, while the state-action value function $Q^{\pi}(s, a)$ does.

Episodic or Infinite Horizon Learning A RL problem can be either episodic or have an infinite horizon. An episodic problem has an end goal. When it is reached the problem is solved, like in a game of chess when the king is captured. A game from start to end is called an *episode*. But some problems continue while receiving rewards, e.g., driving a car. Subgoals should be reached all the time, for example changing lane, but the task continues.

Policies There are multiple possible policies π. A greedy policy is a policy that always takes the action for which the estimated total return is highest;

$$\pi_{greedy} = \arg\max_{a'} Q(s, a') \tag{2.1}$$

When the Q-values are not fully converged yet, this is not optimal. In order to find a better solution than currently found, the agent needs to explore unknown options. To allow exploration, an ϵ-greedy policy can be used:

$$\pi_{\epsilon-greedy} = \begin{cases} \arg\max_{a'} Q(s, a'), & \text{if } p \text{ is } > \epsilon \\ \text{random}(\mathcal{A}), & \text{if } p \text{ is } < \epsilon \end{cases} \tag{2.2}$$

Here p is a random number in [0,1] and ϵ is the exploration rate. When an exploratory action is taken, there is no distinction between a suboptimal action (just not the best) and an action that leads to failure, the agent chooses equally among all actions.

Another action selection method is *Softmax action selection* [12]. With Softmax, the greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their estimated value. The most common form uses the Gibbs distribution:

$$\text{Probability action } a \text{ is chosen} = \frac{e^{Q(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q(s,b)/\tau}} \tag{2.3}$$

The positive parameter τ is called the *temperature*. This parameter is used to tune the behavior of the formula. A high temperature results in nearly equiprobable actions while a low temperature

results in an almost greedy behavior. Here n is the total amount of actions possible in the current state.

2.2 Temporal-Difference Learning

Temporal-Difference methods (TD) do not use a model. They do not postpone the updating of the value function to the end of an episode as do Monte Carlo methods; instead they update during the episode. And where Dynamic Programming methods use all possible follow ups of a state in the update, TD only uses one of the possible follow up states (see [1] for further reading about these methods).
It is most common in TD learning to estimate state-action values, i.e. to estimate the Q-values. There are two ways of estimating the Q-values: on-policy and off-policy. The difference shall be explained in Section 2.2.1.

Sarsa is a widely used form of on-policy TD learning and works as follows:
After an action is chosen and the system went from state s_t to the next state s_{t+1}, the action-value function is updated. The update rule for TD is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta \tag{2.4}$$

The TD-error δ for Sarsa is:

$$\delta \leftarrow r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{2.5}$$

Here α is the learning rate. The TD-error is a measure of the difference in expectation between following states. Because the transition function of the problem is usually stochastic, it is unwise to replace the old estimate with the new one. The **learning rate** α determines the mixing ratio between the new and the old estimate. A high learning rate is beneficial for the learning speed at the beginning of learning and it also facilitates adaptation to changed circumstances in a later stage of the process. But a high learning rate can also lead to an oscillating solution.
The **time discount rate** γ makes sure that state-action pairs that lead to a reward N time steps later, will be rewarded less. To clarify why this is needed the formula of the sum of the discounted rewards is given:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.6}$$

R_t, the total discounted reward, is what the agent tries to maximize. If γ is 1, so there is no discounting, the agent does not differentiate between a reward given now or a reward given far in the future. When dealing with a continuous problem the final time step can be infinity. Without any discounting this would make the total reward R_t also infinite. A discount rate of zero would make the agent only interested in the current reward. It would have no insight in the future whatsoever. Therefore, γ is chosen $0 < \gamma < 1$ with γ close to 1.

2.2.1 On-Policy versus Off-Policy TD

Sarsa is on-policy because the Q-value used to update the current value belongs to the state that is actually visited in the next step. The off-policy method is Q-learning. The update rule for Q-learning differs from Sarsa only in TD-error, which has a max operator: The TD-error δ for Q-learning is:

$$\delta \leftarrow r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{2.7}$$

This method is off-policy because the best following state-action pair is chosen to update the current value with, even if a suboptimal action is chosen in the next step. This ensures that, regardless of the used policy, the solution will converge to the optimal solution. The effect of this difference can be explained by the following example using Figure 2.1. Here a grid world is presented with a cliff in it. The task is to walk from start to goal. Using Q-learning will result in the path close along the edge but Sarsa will not because it will take into account that the agent may take exploratory steps while walking alongside the cliff and fall.

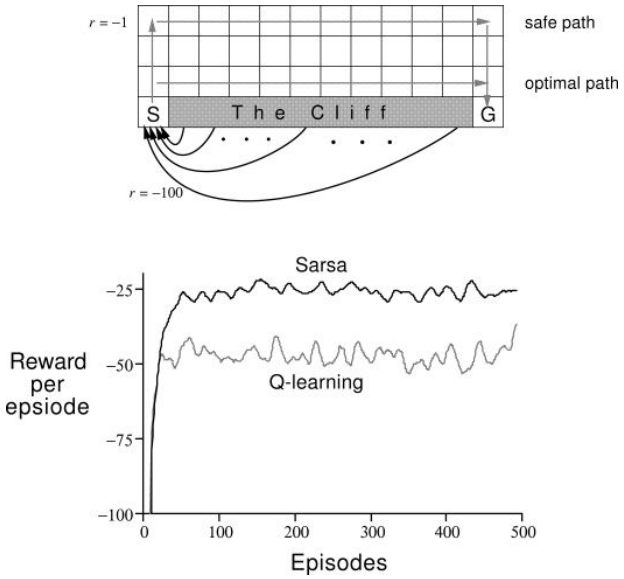


Figure 2.1: *Sarsa and Q-learning [1], figure 6.13. The difference between on-policy and off-policy methods is shown here. Q-learning results in the optimal path. Sarsa takes into account the chance of exploring near the cliff and falling, hence results in the safe path.*

2.2.2 Eligibility Traces

In normal TD learning, only the previous state-action pair is held responsible for the reward given in the next state. Eligibility traces are a way to keep track of the state-action pairs visited and are used to give earlier pairs part of the credit or blame for getting a positive or negative reward. It is important to realize that this speeds up convergence but has no affect on the final solution. For every state-action pair an *eligibility trace* $e_t(s, a)$ is kept. When the state-action pair is visited the eligibility trace is incremented by 1. The eligibility trace is discounted exponentially each time step by the *trace-decay factor* λ times the discount rate γ. Each time step not only the last chosen state-action pair is updated but all pairs with a non-zero eligibility trace.

Q-learning combined with eligibility traces is called $Q(\lambda)$, Sarsa combined with eligibility traces is called Sarsa(λ). All the state-action pairs in the trace are updated every time step according to:

$$Q(s_{t-k}, a_{t-k}) \leftarrow Q(s_{t-k}, a_{t-k}) + \alpha\gamma^k\lambda^k\delta_t \tag{2.8}$$

A problem with eligibility traces occurs when a state is visited a second time before the first eligibility trace has fully decayed. In this case the trace can exceed 1. A solution is to use replacing traces that limit the trace to a maximum of 1. Another problem is the length of the trace, which may cause calculation times to grow. This can be solved by using a threshold for

the smallest relevant eligibility trace, which results in a maximum length of the trace. When an exploratory action is chosen for Q-learning, the eligibility trace is set to zero because the previous actions are not responsible for the current state.

2.3 Generalization

In continuous time problems the value function can not consist of a look-up table with a Q-value for each state-action pair since the number of possible pairs is infinite. Instead, the information is represented by a function of a set of parameters. This is called generalization and it has several advantages. By generalizing the value function, less memory is needed to store it. Generalization also makes it superfluous to visit *every* state of the state-space in order to result in a developed policy. This section is focused on linear function approximation. Common linear function approximation methods are coarse coding, tile coding and radial basis functions. In this research tile coding is used because of its low computational costs, good representational power and ease of use. Tile coding will be explained here.

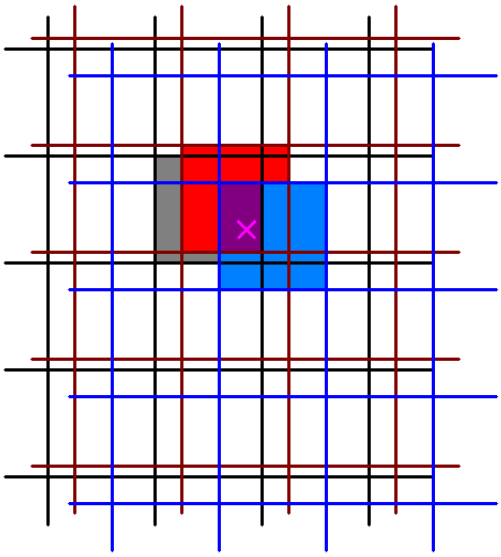


Figure 2.2: *Tile coding. Three tilings are stacked, each with 12 tiles. The cross marks the current state. The coloured tiles mark the tiles containing the current state.*

In tile coding, several grids of tiles are placed on top of each other, as shown if Figure 2.2. The grids are (randomly) shifted with respect to each other so no tiles cover the same regions of the state space. For the shifting algorithm used in this research, see [17]. The values of the tiles are represented by the vector $\vec{\theta_t} = (\theta_t(1), \theta_t(2), \ldots \theta_t(n))^T$, where each entry of $\vec{\theta_t}$ is a value of a tile and the values can change over time. For every state-action pair there is a vector $\vec{\phi}_{s,a}$ that holds a 1 for every tile ($\theta_t(i)$) that contains the state-action pair, and 0 for all tiles that do not. When the value of a pair is requested, the value is calculated by:

$$Q_t(s_t, a_t) = \sum_{i=1}^{n} \theta_t(i)\phi_{s,a}(i) \tag{2.9}$$

where n is the number of tiles. When a new estimate for a state-action pair is available, $\vec{\theta_t}$ is updated. The update rule is as follows:

$$\vec{\theta}_{t+1} = \vec{\theta_t} + \alpha[q_t(s_t, a_t) - Q_t(s_t, a_t)]\vec{\phi}_{s,a} \tag{2.10}$$

Here $q_t(s_t, a_t)$ is the new estimate, and $Q_t(s_t, a_t)$ the old estimate. The step-size parameter α ensures that the values of the tiles are gradually updated by the difference in estimation. A commonly used guideline for the learning rate is: $\alpha = \frac{1}{\#tilings}$. This is necessary because $\vec{\theta_t}$ has a limited amount of parameters and can therefore never represent all values with 100 percent accuracy. With generalization, there is also an estimate available for state-action pairs that have not been visited yet. Also, state-action pairs are influenced by the update of pairs which share the same tiles, this smoothens the value function. Not all value function can be represented because there is a limit to the amount of sharp changes that can be represented with tile coding.

Chapter 3

Inverted Pendulum on a Cart Model

To test the proposed methods, a rather simple test setup is used. This has the advantage that the learning time is short and it is easier to observe what is happening in the system. The model of an inverted pendulum on a cart is chosen, see Figure 3.1. This model and a walker are both dynamic systems and deal with gravity and they are both continuous systems. It is also likely that the failure states for the model of an inverted pendulum on a cart have a similar dispersion over the state-space as the failure states for a walker. The failure states are expected to be concentrated in certain areas.

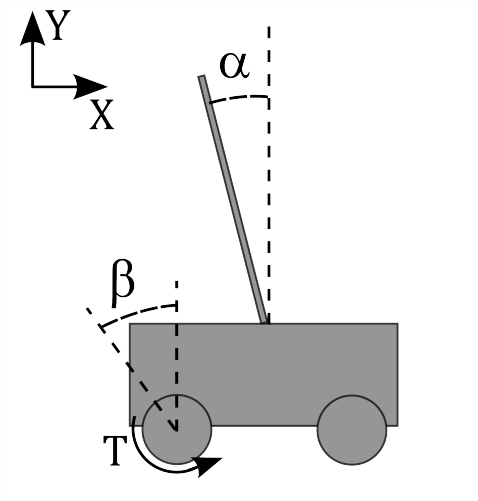


Figure 3.1: *Inverted pendulum on a cart, α is the angle of the pole, β is the angle of the cart's wheels and T is the torque on the wheels.*

3.1 The Learning Problem

Here, the characteristics of the learning problem are given. The state of the inverted pendulum is $\left\{\begin{matrix}\alpha\\ \dot{\alpha}\\ \dot{\beta}\end{matrix}\right\}\left\{\begin{matrix}rad\\ rads^{-1}\\ rads^{-1}\end{matrix}\right\}$ with α the angle of the pole, $\dot{\alpha}$ the angular velocity of the pole and $\dot{\beta}$ is the angular velocity of the cart's wheels. The action of the inverted pendulum is $\{T\}$, the torque on

the wheels. The action space consists of 5 actions; $[\ -1.5 \quad -0.75 \quad 0 \quad 0.75 \quad 1.5\ ]Nm$.

3.1.1 Reward Structure

The reward structure consists of a positive reward for balancing the pole (200) and a negative reward for letting the pole fall (-100). The condition for falling is: $\|\alpha\| \leq 0.45\pi$. The conditions for balancing are: $\{\|\alpha\| \leq 0.005\pi\} \wedge \{\|\dot{\alpha}\| \leq 0.005\pi\}$. No time penalty is added.

After the pendulum fell or met the balancing conditions, the episode ends. Hence, this task is episodic while walking is not. The main measure of performance in this research is falls during learning, because the inverted pendulum does not fall often after reaching the balancing state, the task is ended.

3.1.2 Step time

The step time of the simulation is 5ms.

3.1.3 Representation of the State-Action Value Function

State-action pairs and their Q-values are stored using tile coding, see section 2.3 for an explanation. The number of tiles is 12 per 2π radian of the pole angle, 6 tiles per 2π radian per second of the pole's angular velocity and 6 tiles per 2π radian per second for the cart's wheels angular velocity. The number of tilings is 16. The Q-space is randomly initialized between 0 and 0.1.

3.1.4 Learning Parameters

The discount rate γ is 0.98.
The exploration rate ϵ is 0.05.
The learning rate α is 0.4.
The trace fall off rate is λ 0.92.
These parameters are kept constant during the course of the simulations.

In Figure 4.3 in Section 4.2, the result of learning with Sarsa and ϵ-greedy is shown.

Chapter 4

Threshold Restricted Learning

In this chapter, a new method called Threshold Restricted Learning (TRL) is introduced. TRL is a method for action selection that, during exploration, makes the probability an action is chosen dependent on the expected reward of the state-action pair. Unlike ϵ-greedy, TRL is able to avoid known failure states during exploration. The concept of TRL developed from the desire to optimize the use of a pre-learned Q-function. Hence, TRL is tested in combination with pre-learning. The influence of different system variables on the performance of TRL is shown. A rule of thumb for the implementation of TRL is established.

4.1 Working Principle

Threshold Restricted Learning (TRL) is a method for action selection. It follows from the idea of using a pre-learned Q-function as effectively as possible. Pre-learning is the use of a model to learn a Q-function in simulation and subsequently use this Q-function as initialization on the real system. Models are never completely accurate but it seems logical to think that a large amount of failure states in simulation correspond to actual failure states. TRL makes use of this by changing the rules for exploration. Where ϵ-greedy chooses a random action out of all possible actions when an exploratory step is taken, TRL picks a random action out of a set of actions with Q-values above a certain threshold. If there are no actions available with Q-values above this threshold, a random action out of all actions is chosen. The height of the threshold defines the behavior of the method. A high threshold favors the solutions already found and only explores when no satisfying action is available in the current state. A low threshold only avoids accidentally picking an action with a low Q-value. To ensure that a method converges to a solution, it needs to update all states-action pairs an infinite amount of times. Therefore, there is a small probability implemented that an action below the threshold is chosen, even though there are actions above the threshold available. In the majority of this chapter, TRL is used in combination with pre-learning. During pre-learning itself, ϵ-greedy is used as action selection method. In pseudo-code, taking an exploratory step with TRL looks as follows:

ALGORITHM 1 Exploration Step TRL

```
1    make action lists:  AboveThreshold and BelowThreshold
2    IF (random number ≤ 0.01 || AboveThreshold = EMPTY)
3       action ← take random action (BelowThreshold)
4    ELSE
5       action ← take random action (AboveThreshold)
```

4.1.1 Related Work

TRL has some similarities to Softmax action selection, explained in chapter 2. Both methods make the Q-value of a state-action pair of influence on the probability an action is chosen. To clarify the difference in action selection between ϵ-greedy, Softmax and TRL, the probability of selection of actions per method are shown in Figure 4.1. The X-axis holds the action space for one state, the Y-axis has a different meaning for the red and the blue line. For the blue dotted line in each plot, th Y-axis represents the Q-value of each action. For the red line, the Y-axis represents the probability the action will be chosen.

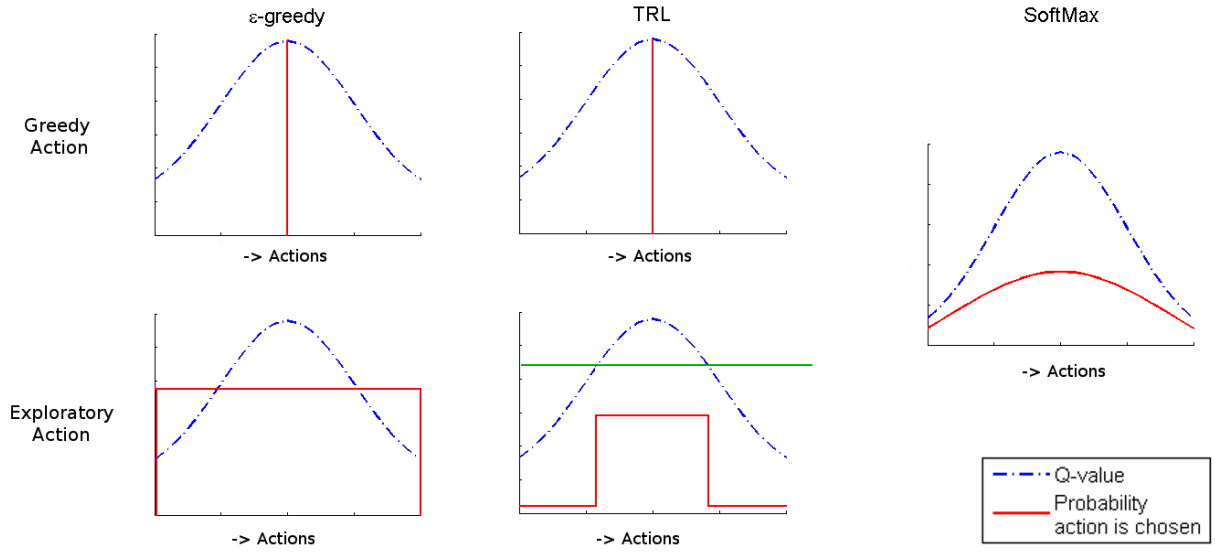


Figure 4.1: *Action selection probabilities and Q-values per method for an exemplar system. In the current state of the system, the Q-values for the actions have a normal distribution. For ϵ-greedy and TRL, the exploration rate defines the probability an exploratory step is taken. Softmax action selection does not have a distinction between greedy and exploratory steps. ϵ-greedy and TRL take the same greedy action, the action with the highest Q-value. They differ in taking an exploratory action. ϵ-greedy takes a random action out of all actions, TRL takes a random action out of the pool of actions with a Q-value higher than the threshold (green line), with a small probability of taking a random action outside that pool.*

An important difference between Softmax and TRL is that Softmax bases the probability of choosing an action on the relative Q-value of that action with respect to the other actions. In contrast, TRL bases the probability of choosing an action on the absolute Q-value of that action. This has two disadvantages.

When the discount rate is smaller than 1, which is the case for the inverted pendulum problem, states-action pairs further away from the goal will have a lower Q-value than state-action pairs near the goal. Adding a time penalty to the reward structure also downgrades Q-values further away from the goal. This gives the threshold a different meaning for different state-action pairs. e.g., a threshold of 50 might have little meaning close to the goal, where all Q-values are above 100, even the ones that occasionally and in failure. Further away from the goal a similar problem occurs. If the current state is distant enough, even the optimal action may not have an expected reward above 50. Because Softmax rates the actions based on the relative expectations it does not produce this problem.
The second disadvantage is the fact that the action selection of TRL is discontinuous. A small change in the Q-value of an action can result in a very large change of the probability that the action will be chosen, which makes the method less predictable.
A large advantage of TRL is that, for Q-values above the threshold, TRL performs a truly exploratory step, Softmax does not. E.g., for Softmax it is impossible to distinguish the difference between a Q-value of 50 and a Q-value of 10, without distinguish the difference between a Q-value of 200 and a Q-value of 170. Next to that, the temperature τ used by Softmax determines the 'greediness' but also determines the effective exploration rate. For Softmax, more exploration is equivalent to 'wilder' exploration. TRL has these parameters separated and can therefore be used to explore more often but keep the same restriction on the exploration.

In order to use TRL in practice, the threshold needs to be estimated beforehand. In the Section Implementation and Results (4.2), TRL is not only shown to work, the influence of several variables, that are inherent features of the system, is also discussed. The goal is to establish a rule of thumb to choose a threshold. In the remainder of this chapter, it is determined if these variables need to be incorporated in the rule of thumb and what their influence on the performance of the method is. The following variables are of interest:

1. Difference between pre-learn simulation and real world
   A large difference between the two systems is expected to make pre-learning in general undesirable, whereas a very small difference is expected to favor high thresholds because more greedy exploration is advantageous.

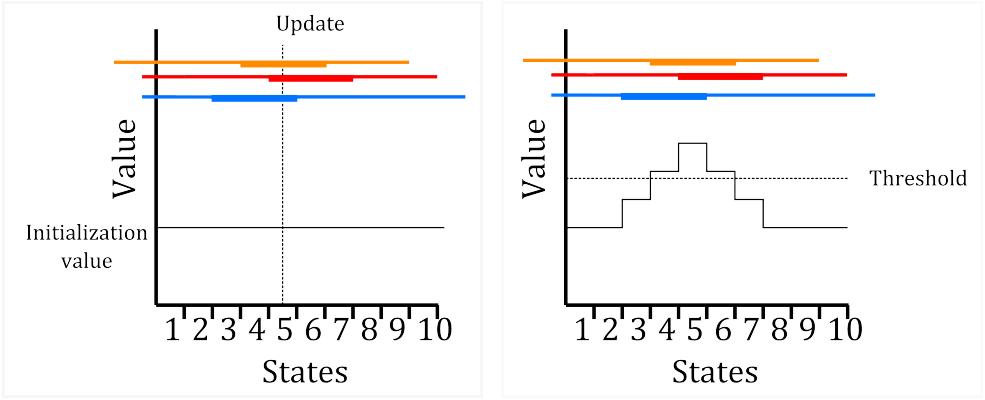
2. The amount of convergence of the pre-learned Q-function
   A completely converged Q-function might be disadvantageous for TRL because the two simulations are not the same. The wrongly learned Q-values might need to be unlearned again. On the other hand, an underdeveloped Q-function may not have enough information in order for TRL to avoid bad action during exploration steps.

3. Exploration rate
   Since TRL only differs from ϵ-greedy when an exploratory step is chosen, it is expected that changing the exploration rate will influence the relative performance of TRL.

4. Discount rate
   As mentioned above, the discount rate influences the optimal threshold for TRL because it determines how much influence current rewards exert on earlier state-action pairs. So a pair will have a different converged expected reward depending on the discount rate. This is expected to influence the optimal threshold.

4.1.2 Effect of Generalization on TRL

Generalization is used to reduce memory size and to speed up convergence, but for TRL it has another advantage. At the start of learning, when little state-action pairs have high expected rewards, exploring with a high threshold is often similar to taking a greedy step. A truly random exploratory step is taken when there are no state-action pairs with a high expectation available, or by a small, build in probability that TRL picks an action below the threshold. Generalization can accelerate the growth of the list of state-action pairs with an expected reward above the threshold. Figure 4.2 shows a state-value function of a discrete system that consists of 1 variable and 10 states. The generalization used to represent the value function is tile coding, see Chapter 2 for an explanation of tile coding. The tiles are 3 states wide and there are 3 tilings. The thick parts of the tilings indicate the tiles that include state 5. After updating state 5, also state 4 and 6 have expected rewards above the threshold. With generalization, the amount of states available for safe exploration can grow faster than without generalization.



(a) *Value function before the update on state 5 is done. All states have the same value. The thick parts of the tilings indicate the tiles that include state 5 and are to be updated.*

(b) *The value function after updating state 5. The values of surrounding states change according to the tiles they share with state 5. The dotted line is the heigth of the threshold.*

Figure 4.2: *The state-value function of a discrete system with 1 variable and 10 states. The value function is represented using tile coding as generalization. The tiles are 3 states wide and there are 3 tilings. After updating state 5, also state 4 and 6 have expected rewards above the threshold.*

4.1.3 Sarsa as Learning Algorithm

In this research, the different action selection algorithms are used in combination with Sarsa. Sarsa is chosen because the method is on-policy and therefore its expectations with respect to exploration are realistic. As opposed to off-policy methods, Sarsa updates a Q-value with the actual following Q-value, not the best following Q-value possible. Sarsa takes into account the possibility of an unfortunate exploratory step. Therefore it is better suited for the avoidance of failure states. See chapter 2 for an explanation of Sarsa.

4.2 Implementation and Results

Sarsa with TRL is tested on the inverted pendulum simulation. The method is tested for several thresholds ranging between the negative and positive reward (-100, 200); -50, 0, 50, 100 and 150.

The chance TRL will choose an action below the threshold, while there are alternatives above the threshold, is 0.01. The influence of the pre-learning simulation, the amount of pre-learning, the exploration rate and the discount rate on the performance are discussed. The approach to test the influence of these variables, and the result of these tests, are described here.

4.2.1 Difference between pre-learn simulation and real world

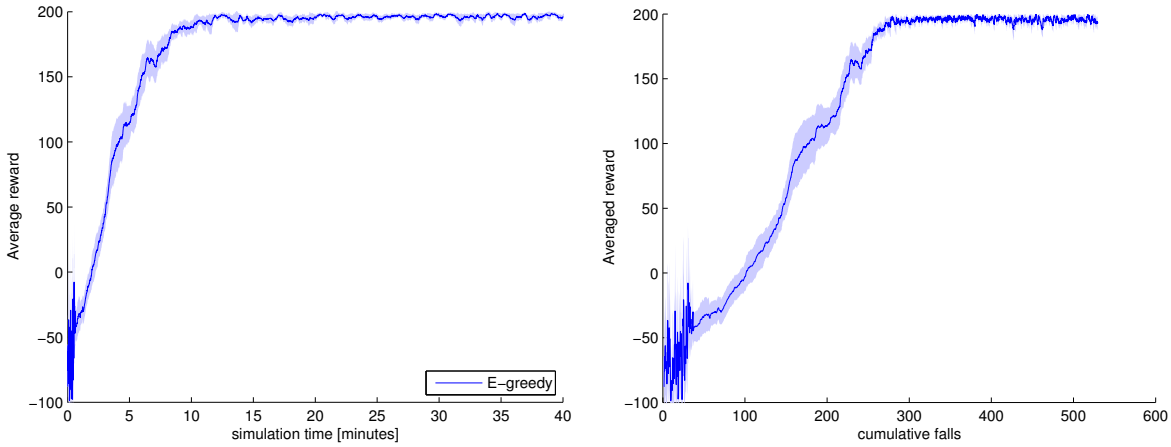
The amount of difference between the two simulations is expected to have a large influence on the performance of TRL. To demonstrate this, 5 scenarios are used for the pre-learning simulation.
To simulate the difference between the model used for pre-learning and the real world, two simulations are used. The first simulation, the pre-learning simulation, represents the simulation model. The second simulation, further referred to as the actual simulation, represents the real world. In this research, two approaches to this difference are used.

Firstly, the difference is modeled as a result of estimation errors. The engineer builds a model for pre-learning and estimates the unknown system parameters. Here, two parameters are chosen to vary in order to mimic the estimation errors. The first one is the dry friction on the joint that connects the pole to the cart. The dry friction is a Coulomb friction and acts as a torque on the joint. The second parameter is the linear damping coefficient for the motors of the wheels. For the linear damping coefficient c holds: $T = c \cdot \omega$. Where T is the torque produced by the damping and ω is the rotational speed on the wheels. Varying these parameters is not necessarily the best way to demonstrate the effect of the method because there is not much freedom to choose these parameters independently. Only a small amount of combinations still results in an inverted pendulum that can be balanced with the given action space.

A second way is to alter the actions of the pole balancer by multiplying it by a scaling factor. This alteration is based on the effect of a rising temperature in the motors. When the temperature rises, the resistance in the wires of the electro motors will rise too, which results in lower torques.

To limit the size of the data, the Q-function of 5 pre-learned lives is saved and is used as initialization of the Q-function for simulation on the actual simulation. The first 8 lives 'in the real world' use the first stored Q-function, the second 8 lives use the second, and so on. 40 lives are simulated on the real world simulation. To show that 5 pre-learned Q-functions is sufficient, after simulation on the actual simulation, the standard deviation of the mean average total reward of each group of 8 lives is calculated and compared to the standard deviation of the mean average total reward of the 40 lives together. The standard deviations of the individual groups are of the same order as the deviation of the group as a whole. This means that the differences between the 5 pre-learned Q-functions do not introduce more variation than the differences already introduced by the lives within one group.

First, the results of learning with Sarsa and ϵ-greedy without any pre-learning is shown so later results can be compared to it.



(a) *Learning curve.*

(b) *Average total reward versus falls. The beginning of the curve is very erratic because there is no moving average of the rewards for the first 80 episodes.*

Figure 4.3: *The learning curve and the average total reward versus falls, without pre-learning on the actual system. These graphs are averaged over 20 lives. On each life a moving average filter is used over 80 episodes. The 95% confidence interval is plotted as the bound. During learning, the system fell on average 290 times.*

In Figure 4.3a the learning curve is shown. On the horizontal axis stands the time in simulation minutes and on the vertical axis the averaged total reward per episode is shown. This average total reward per episode is constructed out of the total rewards at the end of each episode. The moving average of 80 rewards is calculated for each separate life. The average of these 20 averaged lives is plotted as averaged reward in the figure. The error bound shows the error introduced by averaging over the 20 lives, after the moving average filter. The beginning of the curve is very erratic because there is no moving average of the rewards for the first 80 episodes.
In Figure 4.3b, the average reward versus falls is shown. The average reward is plotted versus the amount of falls. The 95% confidence interval is plotted as the bound. In the remainder of this chapter, more methods are plotted together without an error bound, to preserve the readability. The average reward in this plot is also a moving average over 80 episodes. This figure gives a more clear view of the amount of falling experience the different methods need to reach a certain average reward. The plots of the normal learning curve give a slightly biased view because they have time on the X-axis. A method that learns fast (with respect to time) might have many very short episodes that ended in failure. Therefore, learning with a minimal amount of failure is not equivalent to learning in a minimal amount of time.

In Table 4.1, the test scenarios for TRL are listed. The first test is done using realistic parameter deviations as the difference between the two simulations. The second and third scenario have magnified alterations of the damping and the friction. Because of this larger difference, the effect of TRL is more clear. The fourth and the fifth scenario have only an altered action space. This is not a realistic variable to vary, but the effect of the alteration is easier to interpret than that of the previous scenarios. Scenario 6 is of use in Section 4.2.2 because it turns out to be a special case with respect to the amount of convergence of the used Q-function.

Scenario	Linear Damping Coefficient	Dry Friction	Torque Scaling
Actual System	0.08	0	1
1. Realistic Scenario	0.088	0.03	1
2. Half Damping, Low Friction	0.04	0.02	1
3. Low Damping, High Friction	0.01	0.05	1
4. Actions with Increased Power	0.08	0	2.5
5. Actions with Decreased Power	0.08	0	0.5
6. Actions with Increased Power + Realistic Variable Alteration	0.088	0.03	2.5

Table 4.1: *Scenarios for testing TRL. The variables listed for scenarios 1 to 5 are the alterations used for the pre-learning simulations. After pre-learning, TRL is used for learning on the actual system.*

1. Realistic Scenario The 'estimation error' for the linear damping coefficient is 10%, and the dry friction is 2% of the maximal torque of the polebalancer. The differences between the parameters for the two simulations are as follows:

Scenario	Linear Damping Coefficient	Dry Friction	Torque Scaling
Actual System	0.08	0	1
1. Realistic Scenario	0.088	0.03	1

Pre-learning on the pre-learn simulation is done until the average, over 80 episodes, of the total reward is 198. The differences between the pre-learn simulation and the real simulation are too small to show any effect of restricted exploration, as can be seen in the learning curve and the rewards versus falls in Figure 4.4.

2. Half Damping, Low Friction In this scenario the difference between the two simulations is more extreme. It is unlikely that one would make these estimation errors. But it is possible that a real system is used for pre-learning and another, similar, real system is the final system. For example, a bipedal walker with protective padding and a bipedal walker without. Here the differences may be large and not alterable. For this scenario the dry friction is 1.3% of the maximum torque and the linear damping is reduced by 50%.
The differences between the two simulations are:

Scenario	Linear Damping Coefficient	Dry Friction	Torque Scaling
Actual System	0.08	0	1
2. Half Damping, Low Friction	0.04	0.02	1

The learning curve after pre-learning for this second scenario is shown in Figure 4.5a. In Figure

4.5b, the average reward versus falls is shown. Here, TRL 50 learns fastest. To examine if there is a significant difference between the performances of the methods during learning, a definition of the end of the learning time is needed. As a measure of how much the average reward fluctuates over time, the amount of change of the average reward over 250 episodes is taken. This gradient of the average reward is plotted in Figure 4.6a as the thick line. After the learning curve becomes stable around an average reward of 195, occasionally failure occurs and the gradient of the average reward is nonzero. The highest observed disturbance in the gradient is just under 3. Therefore, when the change of the average reward over 250 episodes is less than 3, the method is said to have converged. The dotted line shows the end of learning point. The learning curve is also plotted in Figure 4.6a.

With this definition of the learning time, the amount of falls during learning can be plotted. In Figure 4.6b this average amount of falls is plotted. The 95% confidence interval is drawn as the black vertical line on each bar. No moving average filter is used on the amount of falls during learning.
Here, it is seen that TRL 50 performs approximately 45% better than ϵ-greedy. Because the confidence bounds overlap, a statistical test, ANOVA, is used to determine if this difference is significant. ANOVA calculates the probability that the difference between two results, in this case the amount of falls per life for ϵ-greedy and for TRL, is due to coincidence and not to the difference in the methods. This probability is the p-value. The difference is called significant when this p-value is smaller than 0.01, i.e 1 percent. For the use of the ANOVA test to be justified, the cases must be independent and the variance of the cases should be of the same order and normal. The probability that the difference in the amount of falls between the TRL and ϵ-greedy is coincidental is given for all thresholds in Table 4.2. Only TRL with a threshold of 50 and threshold 100 differ significantly from ϵ-greedy. In the falls during learning plots, a star over a bar is used to indicate the result is significant.

Threshold	p-value
-50	8.5e-1
0	2.4e-2
50	1.4e-5
100	3e-3
150	1.2e-1

Table 4.2: *p-values for falls during learning with respect to ϵ-greedy, scenario 2. Only for TRL 50 and 100 the probability the better result is due to coincidence is smaller than 1%.*

3. Low Damping, High Friction For scenario 3 the dry friction is take 3.3% of the maximum torque and the linear damping is reduced by 12.5%. The differences between the simulations for scenario 3 are as follows:

Scenario	Linear Damping Coefficient	Dry Friction	Torque Scaling
Actual System	0.08	0	1
3. Low Damping, High Friction	0.01	0.05	1

The average reward versus falls and the falls during learning are shown in Figure 4.7. Again TRL 50 provides a reduction on the amount of falls during learning of approximately 45%. Only the results for TRL 50 and 100 are significantly different from ϵ-greedy with, respectivily, a p-value of $7.1e^{-9}$ and a p-value of $1.2e^{-3}$.

The following two scenarios are not based on potential error made during the definition of the model of the system. The only difference between the real system simulation and the pre-learn simulation is a scaling factor on the torque exerted on the pole by the cart.

4. Actions with Increased Power In this scenario only the actions are altered by scaling the torques by a factor of 2.5.

Scenario	Linear Damping Coefficient	Dry Friction	Torque Scaling
Actual System	0.08	0	1
4. Actions with Increased Power	0.08	0	2.5

The average reward versus falls and the falls during learning are shown in Figure 4.8. TRL 50 performs about 11% better. Only the result for TRL 50 is significant with a p-value of $2.1e^{-3}$. Although using this pre-learned Q-function still results in faster learning and learning with less falls, this pre-learn scenario is far from optimal because the total amount of falls is much higher than for the other scenarios.

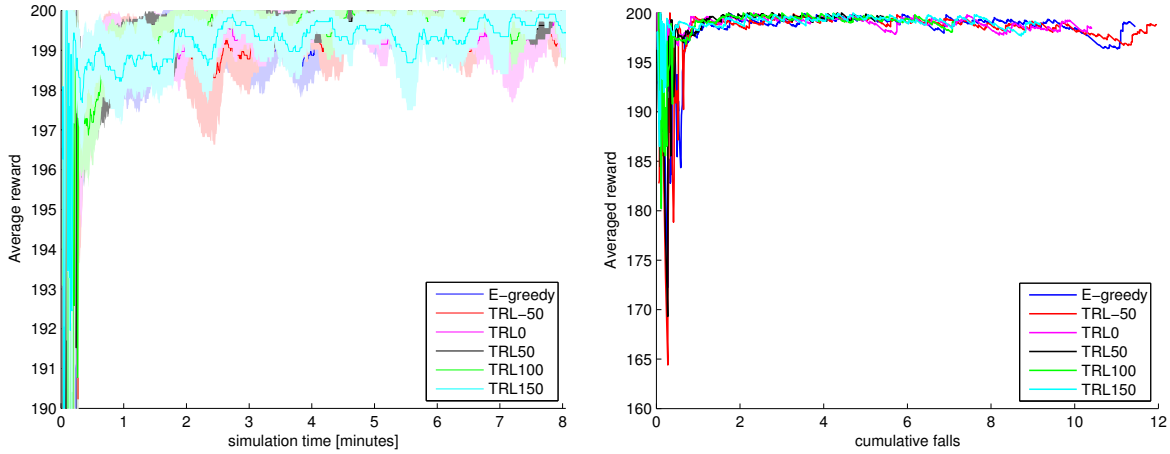
5. Actions with Decreased Power In this scenario, the actions are altered by a scaling factor on the torques of 0.5.

Scenario	Linear Damping Coefficient	Dry Friction	Torque Scaling
Actual System	0.08	0	1
5. Actions with Decreased Power	0.08	0	0.5

The average reward versus falls and the falls during learning are shown in Figure 4.9. The differences between the methods are not significant. But here an interesting effect can be observed. The only difference between scenarios 4, 5 and the actual simulation is the altered torque scaling factor. Pre-learning with scenario 5 results in approximately 10 times less falls than pre-learning with scenario 4. Hence, it can be concluded that pre-learning with actions with less power is less damaging to the result than pre-learning with magnified actions.

TRL Without Pre-learning The idea of TRL came from the desire to effectively use the information pre-learned in simulation. But in fact, TRL is only a method for action selection and hence can be used without pre-learning. Next to that, initialization of the Q-function between 0 and 0.1 can also be seen as the result of learning on a pre-learn simulation that is very far off. In Figure 4.10, Sarsa with ϵ-greedy and Sarsa with TRL are shown for the actual system, without pre-learning. TRL 50 performs about 7% better than ϵ-greedy, with a p-value of $1.3e^{-3}$. TRL 25 performs approximately 6% better, with a p-value of $5.6e^{-3}$.

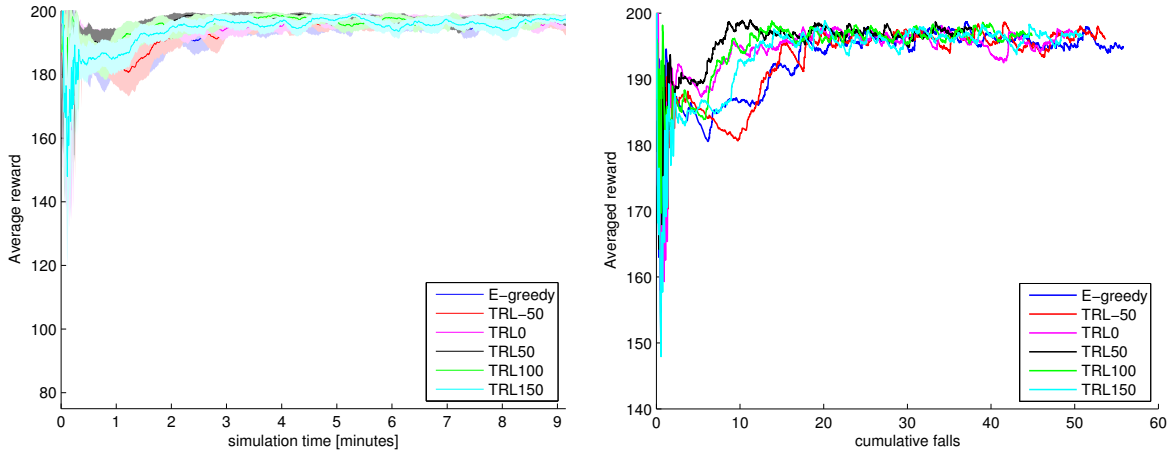
Conclusion for the Difference in Simulations TRL performs better in all but scenario 5, given the right threshold is chosen. But only unrealistically large difference show the benefit of TRL. With pre-learning, the learning is always faster than without. The best threshold for this inverted pendulum case is 50. This does not change over the pre-learn scenarios shown here. The difference between the pre-learn scenarios is not large enough to show a difference in the optimal threshold. Pre-learning with actions with less power is less damaging to the result than pre-learning with magnified actions. TRL without pre-learning also reduces the amount of falls, but by a very small amount.



(a) *Learning curve after pre-learning, the 95% confidence interval is plotted as the bound.*

(b) *Average reward versus falls*

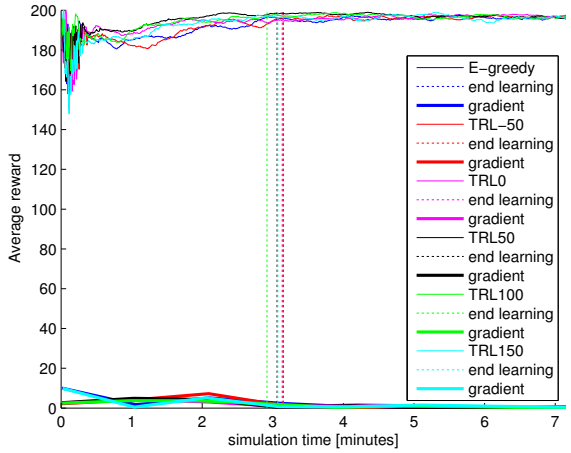
Figure 4.4: *Learning curve after pre-learning and the reward versus falls, scenario 1. Realistic Scenario. No significant difference in performance can be seen between the methods.*



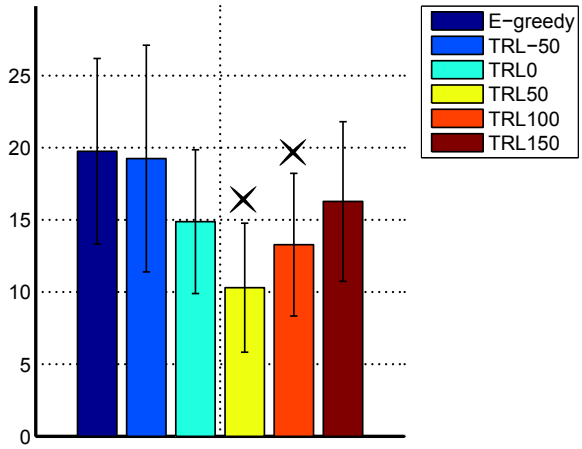
(a) *Learning curve after pre-learning, the 95% confidence interval is plotted as the bound.*

(b) *Average reward versus falls.*

Figure 4.5: *Learning curve after pre-learning and reward versus falls, scenario 2. Half Damping, Low Friction. There is a small difference in learning speed and in the amount of failure before a stable average reward is reached. Whether TRL makes a significant difference can not be concluded from these plots.*

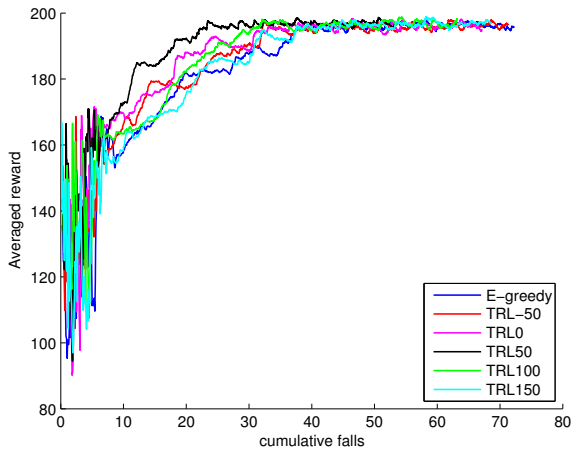


(a) *End of learning. When the gradient of the learning curve, the thick line, is lower than 3, the learning is said to have ended. The dotted vertical line marks the end of learning.*

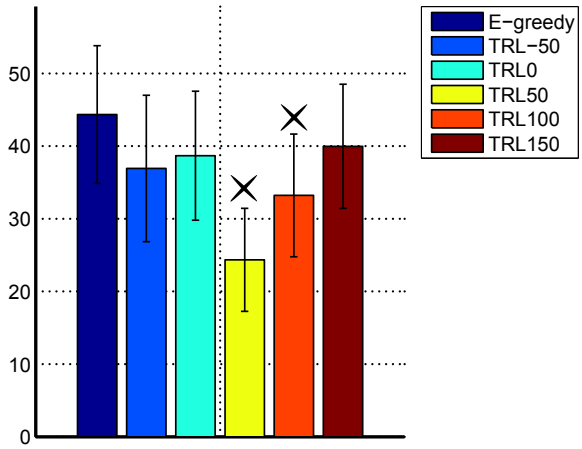


(b) *Amount of falls during learning. This is the amount of falls that occurred before the end of learning as defined in the end of learning curve. Significant results are marked with an X.*

Figure 4.6: *Result for scenario 2. Half Damping, Low Friction. TRL 50 and 100 differ significantly from ϵ-greedy. TRL 50 performs approximately 45% better.*

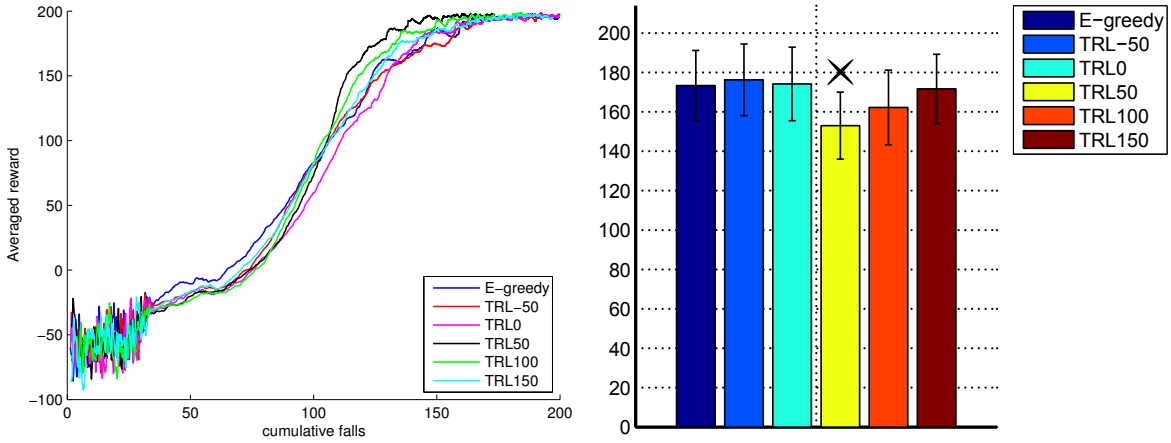


(a) *Average reward versus falls*



(b) *Falls during learning. Significant results are marked with an X.*

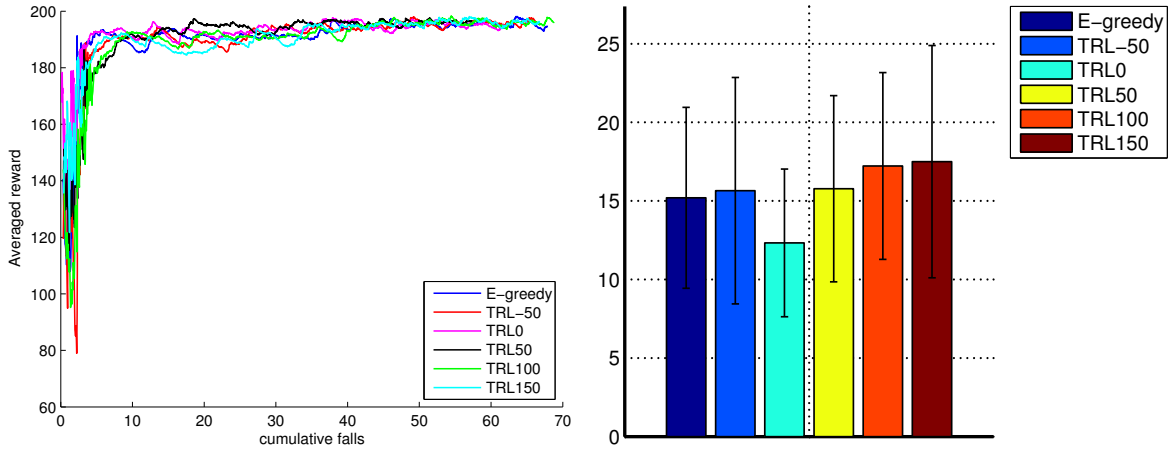
Figure 4.7: *Average reward versus falls and falls during learning, scenario 3. Low Damping, High Friction. TRL 50 falls about 45% less than ϵ-greedy, with a p-value of* $7.1e^{-9}$.



(a) *Average reward versus falls.*

(b) *Falls during learning. Significant results are marked with an X.*

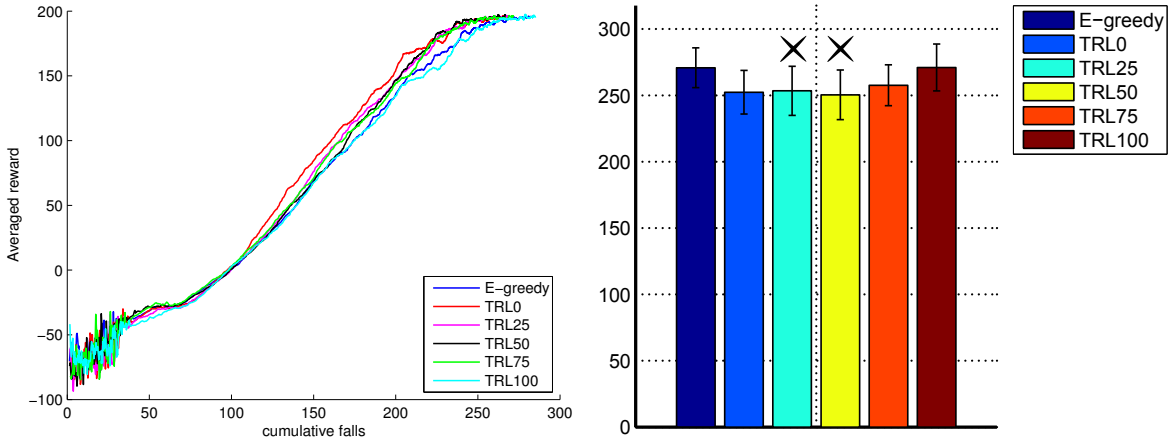
Figure 4.8: *Average reward versus falls and falls during learning, scenario 4. Actions with Increased Power. Only TRL 50 performs significantly better than ϵ-greedy. Pre-learning with this scenario results in only a slightly better performance than without pre-learning.*



(a) *Average reward versus falls.*

(b) *Falls during learning. Significant results are marked with an X.*

Figure 4.9: *Average reward versus falls and falls during learning, scenario 5. Actions with Decreased Power. The results do not differ significantly. But using actions with less power during pre-learning results in better performance than using magnified action.*



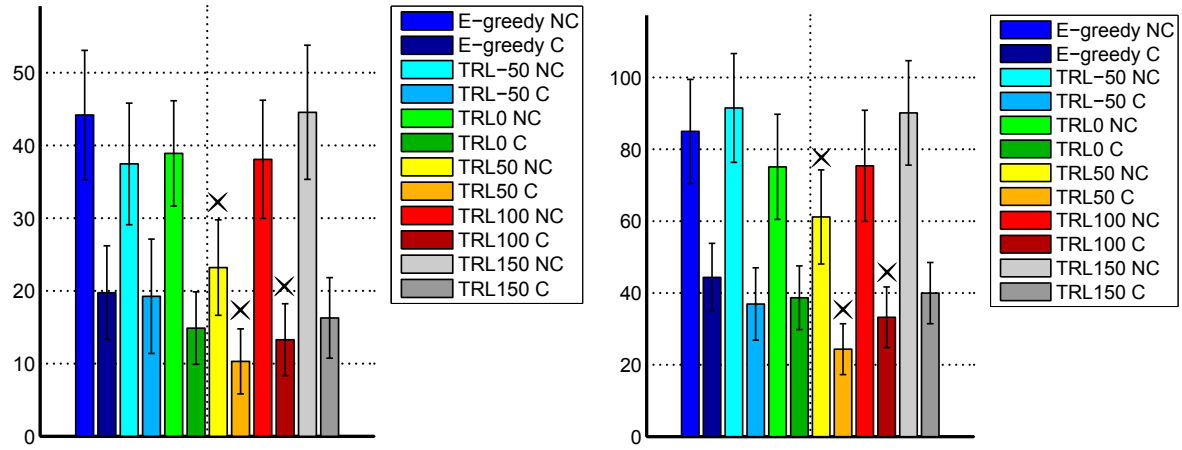
(a) *The average reward versus the amount of falls for the actual simulation, without pre-learning.*

(b) *Falls during learning for the actual simulation, without pre-learning. Significant results are marked with an X.*

Figure 4.10: *Results for the actual simulation without pre-learning. TRL also performs significantly better without pre-learning for a threshold of 25 and 50.*

4.2.2 The amount of convergence of the pre-learned Q-function

To demonstrate the influence of the amount of convergence of the pre-learned Q-function, simulation shall be done using a converged and a non-converged Q-function. For the converged Q-function, pre-learning on the first simulation is done until the average, over 80 episodes, of the total reward is 198. (As mentioned in Chapter 3, the absolute maximum is 200.) For the non-converged Q-function, pre-learning is stopped earlier, at an average total reward of 160.



(a) *Falls during learning for scenario 2. Half Damping, Low Friction, with both a converged and a non-converged pre-learned Q-function (converged data from Figure 4.6b). In the legend, NC stands for non-converged, C for converged. Significant results are marked with an X.*

(b) *Falls during learning with both a converged and a non-converged pre-learned Q-function for scenario 3. Low Damping, High Friction. (converged data from Figure 4.7b). In the legend, NC stands for non-converged, C for converged. Significant results are marked with an X.*

Figure 4.11: *Falls during learning with a non-converged pre-learned Q-function compared to the falls during learning with a converged pre-learned Q-function for scenario 2 and 3. Both scenarios show that a converged Q-function performs approximately 50% better.*

In Figure 4.11a, the falls during learning for scenario 2, after shorter pre-learning, are plotted. From Figure 4.11a it can be seen that with the less converged Q-function all methods perform about 50% worse than with a converged Q-function. But the best performing threshold is still 50 and performs approximately 45% better than ϵ-greedy with a p-value of $2e^{-10}$. The results for the other thresholds are not significant.

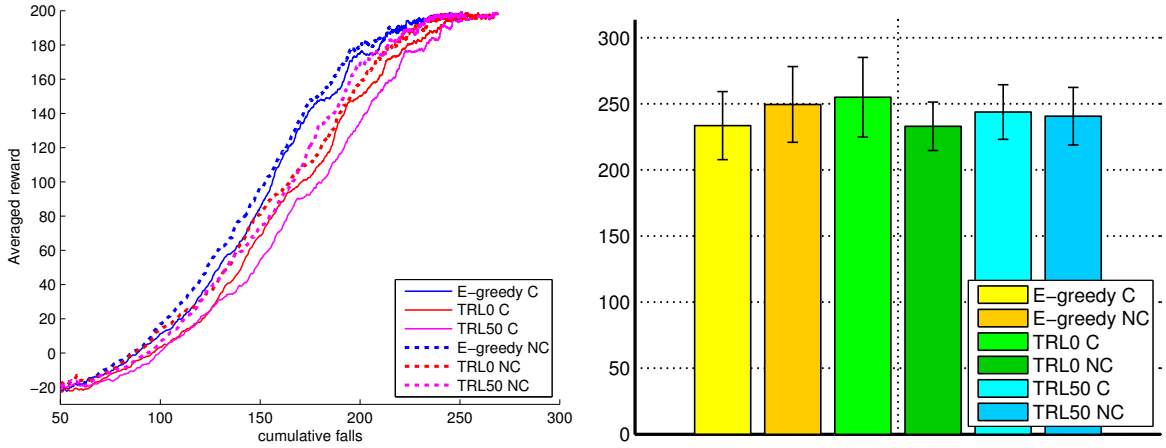
Simulating with a shorter pre-learning time for scenario 3 has a similar result as it did for scenario 2. All methods perform worse, but TRL 50 is still the best threshold and performs significantly better than ϵ-greedy, about 30%, with a p-value of $1.2e^{-5}$. See Figure 4.11b. Again, only the result for TRL 50 is significantly better than ϵ-greedy.

The results for scenario 4 and 5 with shorter pre-learning are similar; all methods perform worse but the optimal threshold is the same and TRL performs significantly better than ϵ-greedy.

A new scenario, scenario 6, is introduced. It has the same linear damping coefficient and dry friction as scenario 1. the Realistic Scenario. But it also has a scaling factor of the torque of 2.5. This scenario shows it is not universally true that a converged pre-learned Q-function performs better than a less converged pre-learned Q-function. With a large enough difference between the pre-learn simulation and the actual simulation, a converged pre-learned Q-function might not perform better, as is the case for scenario 6.

Scenario	Linear Damping Coefficient	Dry Friction	Torque Scaling
Actual System	0.08	0	1
6. Actions with Increased Power + Realistic Variable Alteration	0.088	0.03	2.5

In Figure 4.12, the average total reward versus the amount of falls is shown for ϵ-greedy, TRL 0 and TRL 50. The dotted lines represent the non-converged counterparts of the continuous lines. Though the performance of the non-converged Q-functions is not significantly better, it does refute the idea that a converged Q-function is always preferable. It is important to notice that these results are only slightly better than the result for no pre-learning. The average amount of falls during learning without pre-learning was 290.



(a) *Average total reward versus the amount of falls with converged (C) and non-converged (NC) pre-learned Q-function.*

(b) *Falls during learning with both a converged and a non-converged pre-learned Q-function. In the legend, NC stands for non-converged, C for converged. All shown amount of falls do not differ significantly.*

Figure 4.12: *Results for learning with a converged (C) and non-converged (NC) Q-function. Pre-learned with scenario 6. Actions with Increased Power + Realistic Variable Alteration. For this scenario, a converged pre-learned Q-function does not performs better.*

Conclusion for the amount of convergence of the Q-function Almost all scenarios presented here benefit from a converged pre-learned Q-function as apposed to a less converged pre-learned Q-function. But scenario 6 showed that a converged Q-function does not necessarily perform better than a less converged. This might be caused by the large difference between the pre-learn and actual simulation. This difference can result in two optimal Q-functions that have so little in common, that using one as the initialization of the other is a nuisance. The amount of convergence has no influence on which threshold is best.

4.2.3 The Amount of Exploration

Since TRL only differs from ϵ-greedy when an exploratory step is taken, it is an obvious choice to test how the performance changes when the exploration rate is altered. Therefore the exploration rate is doubled, from 0.05 to 0.1. The result for scenario 2 is shown in Figure 4.13. The performance of all methods is less than with an exploration rate of 0.05, but the difference in performance is larger. TRL 50 performs about 60% better than ϵ-greedy with a p-value smaller than 5^{-11}. All other scenarios have a similar result.

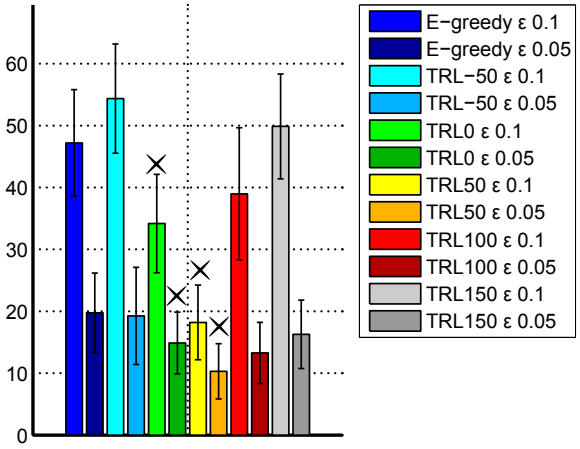


Figure 4.13: *Falls during learning for an exploration rate (ϵ) of 0.1 for scenario 2. Half Damping, Low Friction. The data from Figure 4.6b, where the exploration rate is 0.05, is also plotted in this graph. All methods perform less with a higher exploration rate on the inverted pendulum. TRL 0 and 50 perform significantly better than ϵ-greedy. Significant results are marked with an X.*

Conclusion for the Amount of Exploration From the results the following conclusion can be drawn; the best threshold does not change with a higher exploration rate, the benefit of using TRL becomes larger and the absolute performance of all methods is worse than with the lower exploration rate. But this last statement is problem dependent. The amount of exploration needed is dependent on the difference between the simulation and the real world, and on the system itself.

4.2.4 The Discount Rate

The discount rate influences the performance of TRL and the threshold that is best because it determines how much influence the current rewards exert on earlier state-action pairs. To show the influence of the discount rate on the performance of TRL, the discount rate is varied for the pre-learn simulation **and** the actual simulation for scenario 2.

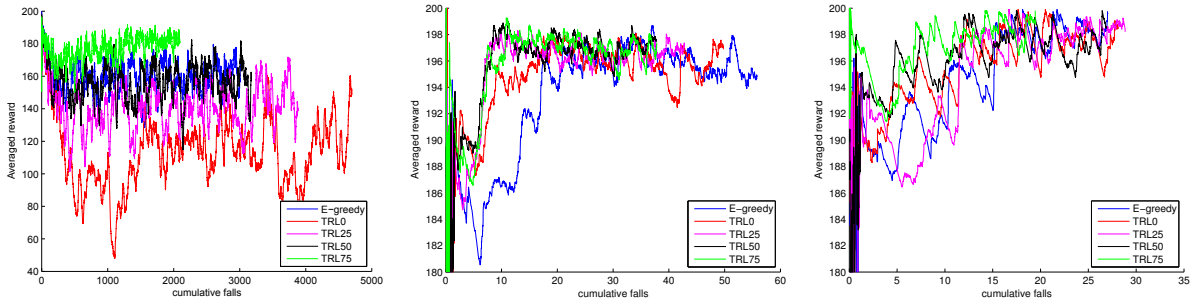
Reducing the discount rate leads to lower expected total rewards, because future rewards are discounted. The share of the goal reward in the update of a state-action pair at a distance of 50 transitions from the goal is: $\gamma^{50} R_{goal}$
For $\gamma = 0.98$ and $R_{goal} = 200$, this share is: $0.98^{50} \cdot 200 = 72.8$
For $\gamma = 0.96$ and $R_{goal} = 200$, this share is: $0.96^{50} \cdot 200 = 26$
For $\gamma = 0.99$ and $R_{goal} = 200$, this share is: $0.99^{50} \cdot 200 = 121$

Because of this, it is expected that the optimal threshold shall decline with a lower discount rate. In Figure 4.14, the learning curve for scenario 2 is plotted for a discount rate of 0.96, 0.98 and 0.99. The simulation plotted in Figure 4.14a is 9 times longer than the others in order to see if the average reward becomes stable. These figures do not support the theorem that a lower discount rate leads to a lower threshold. There are two possible explanations found.

Firstly, on the inverted pendulum example, episodes that end in failure have on average more state transitions than episodes that end in success. This is true for all simulations shown. But the effect of this difference depends on the discount rate. E.g., for the simulation of scenario 2 with a discount rate of 0.98, TRL 50 has failure episodes of, on average, 81 transitions and successful episodes of, on average, 50 transitions. The size of the influence on the update on the first state-action pair for a failure episode is $0.98^{81} = 0.1947$. For a successful episode this is $0.98^{50} = 0.3642$. So the influence of the successful episode is about 2 times bigger than the influence of the failure episode. When the discount rate is taken to be 0.96 and the same episode lengths are used, these values become: $0.96^{81} = 0.0366$ and $0.96^{50} = 0.1299$. Now, the influence of a successful episode is about 3.5 times bigger than the influence of a failure episode. The lower discount rate amplifies the effect of the differences in episode length and this might lead to a positive bias of the expected total rewards.

An explanation for the result shown in Figure 4.14a, with a discount rate of 0.96, is that the discount rate is too low for the experiment. It can be seen that all methods start by unlearning, the average reward declines. The reason for this unlearning is that the average episode length on the pre-learn simulation is shorter than the average episode length on the actual simulation. A possible cause is the lower linear damping on the moters in the pre-learn scenario. The discount rate of 0.96 is too low for the actual system but not for the pre-learn scenario. And because higher thresholds favor short episodes, a threshold of 0 is affected by it the most and a threshold of 75 performs best.

It is very difficult to show the effect of the discount rate on the optimal threshold. But from the brief calculations it can be concluded that the discount rate has a large influence.



(a) *The average reward versus the amount of falls for scenario 2, γ=0.96. The lives of this simulation are 9 times longer than the lives of the other simulations in order to see if the average reward becomes stable.*

(b) *The average reward versus the amount of falls for scenario 2, γ=0.98. This is the γ used in all other tests.*

(c) *The average reward versus the amount of falls for scenario 2, γ=0.99.*

Figure 4.14: *Reward versus falls after pre-learning on scenario 2. The discount rate is altered for the pre-learn simulation and the actual simulation. While it is expected that a lower discount rate leads to a lower optimal threshold, these graphs show that that is not the case.*

4.3 Rule of Thumb for Determining the Optimal Threshold

In order for TRL to be useful in practice, an estimation of the optimal threshold needs to be made beforehand. As is shown in the previous sections, of the four tested variables, only the discount rate has a influence on which threshold is best. But in this section, also the influence of the difference between the two simulations shall be discussed.

A possible approach to come to a good threshold is to calculate the expected rewards that state-action pairs at the beginning of a successful episode will have. For an average successful episode of 55 states, the first state is updated with the goal reward by $0.98^{55} * 200 = 65$, for $\gamma = 0.98$ and the positive reward = 200. A higher threshold than 65 will exclude these state-action pairs while they lead to success. A lower threshold will include pairs with lower expected total rewards. Most of these lower expected total rewards are below 65 because the pair receives a negative reward in the future, with the exception of pairs in the beginning of an episode longer than 55 transitions. The expected episode length is in its turn influenced by the chosen threshold. A higher threshold favors short episodes. This equilibrium complicates the predicting of the optimal threshold. The following rule of thumb is introduced:

$$\text{Thres}_{opt} = \gamma^{n^*} * R_{goal} \tag{4.1}$$

Here Thres_{opt} is the estimated optimal threshold, γ is the discount rate, n^* is the expected average amount of transitions per *successful* episode *during learning*, on the *actual simulation*, *after pre-learning* (if any) and R_{goal} is the positive end reward, in the case of the inverted pendulum this is the reward for balancing. This rule of thumb estimates the Q-value of the first state-action pair of a successful episode during the learning stage.
Here, the expected average amount of transitions per successful episode, n^*, is determined for episodes during the learning stage because that is the stage where TRL needs to perform. When learning is (almost) complete, the episodes are shorter and therefore a higher threshold is optimal

after the learning stage. The used pre-learn simulation partly determines the episode length during learning on the actual simulation. The requirements for Equation 4.1 to be applicable are:

(a) The problem has only end rewards, no time penalty. In Section 4.5, Future Work, a way to integrate the time penalty is suggested.

(b) The learning problem is episodic. It is expected that the R_{goal} can be substituted by the highest intermediate reward, but further research is needed to prove this.

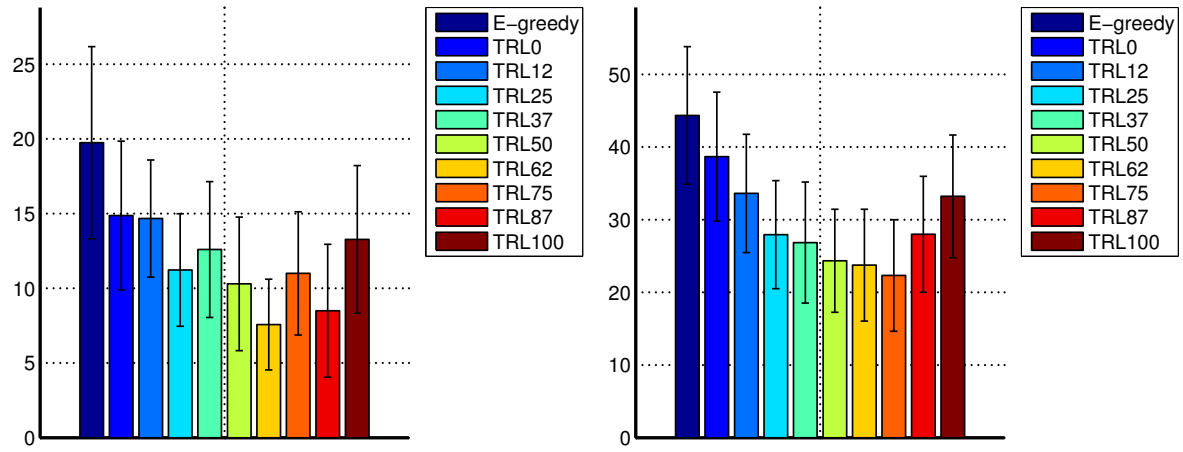
(c) Penalties are below the initialization values of the Q-function in the case no pre-learning is used.

In the case no pre-learning is used, estimating n^* is done in the same manner as estimating γ.

Validation To validate the relation the rule of thumb is based on, test are done with a known number of average amount of transitions per successful episode during learning.
After pre-learning with scenario 2, the average amount of transitions per successful episode during learning, for Sarsa with ϵ-greedy on the actual system, is 57.5. The learning time is defined as 3 simulation minutes for this case, see Figure 4.6. The estimated optimal threshold is calculated to be 62.6. To see if this estimation is reliable, the simulation of the actual system is done for several more thresholds between 0 and 100, see Figure 4.15a. A threshold of 62 is indeed best. Although the difference between TRL 62 and the surrounding thresholds is not significant, the rule of thumb does result in a threshold that performs significantly better than ϵ-greedy.

For scenario 3, the average amount of transitions per successful episode during learning is 59.9. The estimated threshold is 59.6. The results for different thresholds for scenario 3 are shown in Figure 4.15b. A threshold of 75 seems to work best, but the difference with the result for threshold 62 is not significant.

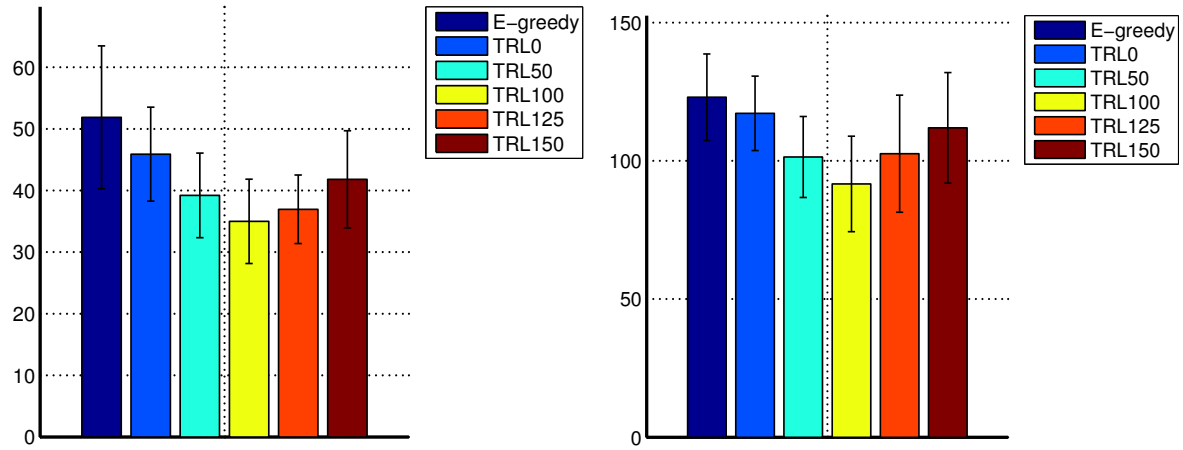
To check if the rule holds, a simulation with scenario 2 is done with a reward for balancing of 400 instead of 200. Here, the average episode length is 61 and the optimal threshold follows to be 115.5. This is in agreement with Figure 4.16a. Scenario 3, with a reward for balancing of 400 is estimated to have the optimal threshold at 103. This is in accordance with Figure 4.16b.



(a) *Falls during learning, scenario 2. The estimated optimal threshold is 62.6.*

(b) *Falls during learning, scenario 3. The estimated optimal threshold is 59.6.*

Figure 4.15: *Falls during learning, scenario 2 and scenario 3. The rule of thumb proves to give good estimations.*



(a) *Falls during learning, scenario 2, reward for balancing = 400. The estimated optimal threshold is 115.5.*

(b) *Falls during learning, scenario 3, reward for balancing = 400. The estimated optimal threshold is 103.*

Figure 4.16: *Falls during learning, scenario 2 and scenario 3, reward for balancing = 400. Again, the rule of thumb proves to give good estimations.*

4.3.1 Using the Rule of Thumb after Pre-Learning

When pre-learning is used, estimating n^* becomes more difficult because the amount of difference between the pre-learn simulation and the real world does not only influence the performance, it also determines the amount of transitions per episode during learning on the actual simulation. And therefore it influences the optimal threshold. The best threshold, after pre-learning on scenarios 2 and 3, was approximately 60, see Figure 4.15. The best threshold without any pre-learning is lower, see Figure 4.10. The estimated best threshold without pre-learning is 42. The n^* used for this estimation was obtained from the simulation. This research does not focus on obtaining a good estimation of the amount of transitions. Estimating the amount of transitions during learning *after pre-learning* is even more difficult since the exact amount of difference between the simulations is not known (if it was known, the pre-learn simulation would be altered to represent the real world perfectly).

Although it did not show in the test on the inverted pendulum, the amount of convergence of the pre-learned Q-function is also of importance. A Q-function that is developed until an average reward of -60 is obtained is an extreme example. In that case, the performance on the actual system shall be virtually the same as without pre-learning.

However, an upper limit for the threshold can be set by using the rule of thumb on the actual system, without pre-learning. In this case, n^* is not the amount of transitions per successful episode during learning, it is the amount of transitions per successful episode after convergence. Episodes are not expected to be shorter than this at any point. The upper limit for the optimal threshold for the inverted pendulum is 72. This is calculated using the rule of thumb and the n^* specified above. In this case, n^* is obtained from simulation. To illustrate that a higher threshold is never optimal, suppose that the Q-function is fully converged and the threshold is 150. With a γ of 0.98 and a reward of 200 at the goal, a state-action pair 50 state transition removed from the goal can reach, at maximum, a Q-value of $0.98^{50} * 200 = 72.8$. Instead of choosing the action leading closer to the goal in this state, TRL 150 takes a random action.

Also as a lower limit, the rule of thumb for the actual system without pre-learning can be used.

Here, n^* is the amount of transitions per successful episode during learning. This lower limit is valid only when the pre-learning Q-function does not deteriorate the performance compared to initialization of the Q-function between 0 and 0.1. Learning episodes are never expected to be longer than these episodes. For the inverted pendulum the lower limit is 42. Which is the answer to everything anyway [18].

4.4 Conclusion

Threshold restricted learning has shown to avoid failure states, on the inverted pendulum simulation, more effectively than ϵ-greedy. To be of use in practice, a good threshold for TRL needs to be estimated beforehand. Four variables were varied in order to determine their influence on the best threshold and their influence on the performance of the method.

The difference between the pre-learn simulation and the actual simulation influences the performance of TRL. A large difference results in inferior performance. For the presented scenarios however, it has no *measurable* influence on the best threshold. In the case of the inverted pendulum as described in Chapter 3, only unrealistic differences make the effect of TRL visible. TRL has proven to also reduce the amount of falls during learning when no pre-learning occurred.

Almost all scenarios presented here benefit from a converged pre-learned Q-function as apposed to a less converged pre-learned Q-function. But when the difference between the pre-learn and actual simulation is too large, the degree of convergence of the pre-learned Q-function is of no importance. This variable has no *measurable* influence on which threshold performs best.

Because TRL only differs from ϵ-greedy when an exploratory step is taken, the gain of using TRL becomes larger when the exploration rate is higher. The exploration rate has no influence on the best threshold.

The discount rate has a large influence on the threshold and on the episode length. It is very difficult to show the effect of the discount rate on the optimal threshold. But from the brief calculations it can be concluded that the discount rate has a large influence.

A relation between the amount of transitions per successful episode during learning and the best threshold is proposed:

$$\text{Thres}_{opt} = \gamma^{n^*} * R_{goal} \tag{4.2}$$

Here Thres_{opt} is the estimated optimal threshold, γ is the discount rate, n^* is the expected average amount of transitions per *successful* episode *during learning*, on the *actual simulation*, *after pre-learning* (if any) and R_{goal} is the positive end reward. This rule of thumb estimates the Q-value of the first state-action pair of a successful episode during the learning stage.

The Applicability of TRL TRL has proven to reduce the amount of falls during learning. With pre-learning, TRL has accomplished a reduction of 50% is some cases. Without pre-learning this is only 7%. However, on the inverted pendulum, the amount of difference between the pre-learn and the actual simulation needs to be large in order to obtain significant results.

The relation between the amount of transitions per successful episode during learning and the best threshold can be used to estimate a good threshold to use on the actual system after pre-learning.

4.5 Future Work

In this research, TRL is tested on a fairly simple inverted pendulum model. In order to see any difference between the methods, the difference between the pre-learn simulation and the actual simulation were too large to be realistic, no engineer would make these estimation errors for the variables. But on a more complex system, small estimation errors might make a difference large enough for TRL to perform significantly better. Therefore, TRL should be tested on a more complex system.

Based on the results of scenario 4 and 5, which led to the conclusion that pre-learning with actions with less power is less damaging to the result than pre-learning with magnified actions,

the question rises if learning with downscaled actions is generally better.

The rule of thumb estimates the Q-value of the first state-action pair of a successful episode during the learning stage. The proposed rule of thumb does not include all possible rewards. For example, when at each time step a $R_{timepenalty}$ is given, the rule of thumb is altered as follows. A penalty given 3 steps ahead (in the future) will have an effect of $\gamma^3 * R_{timepenalty}$. Hence, it is expected that the following alteration to Equation 4.1 will give a good estimation for the threshold for a system using time penalties:

$$\text{Thres}_{opt} = \gamma^{n^*} * R_{goal} + \sum_{k=1}^{n^*} \gamma^k * R_{timepenalty} \tag{4.3}$$

In the same manner, other rewards can be incorporated.

The rule of thumb is proven for an episodic tasks, while walking is a task with an infinite horizon. It is expected that by substituting R_{goal} by the highest intermediate reward, the rule of thumb will hold for infinite horizon tasks.

Chapter 5

Softmax Action Selection

As mentioned in Chapter 4, Softmax action selection and TRL are somewhat similar methods. Hence it is obvious to research if Softmax also has a positive influence on the avoidance of failure states. In Chapter 4 it was explained that Softmax action selection and TRL both have a larger chance to avoid failure states because Softmax never takes a complete random action and TRL tries to minimize it. The working principle of Softmax action selection is explained in Chapter 2.

SoftMax is tested under the same conditions as TRL, and on the same scenarios, see Table 4.1. The pre-learned Q-functions are used as initialization again and SoftMax is used on the actual simulation. The temperatures chosen to test Softmax with are 4, 2, 1 , 0.75 and 0.5. In Figure 5.1, the falls versus rewards are plotted for scenario 2 with a converged Q-function, with temperature values of 4, 6 and 10. Based on this result, temperatures higher than 4 will not be tested. Temperatures lower than 0.5 will not be tested because this results in calculation problems, as the denominator reaches infinity, see formula 2.3 in Chapter 2.

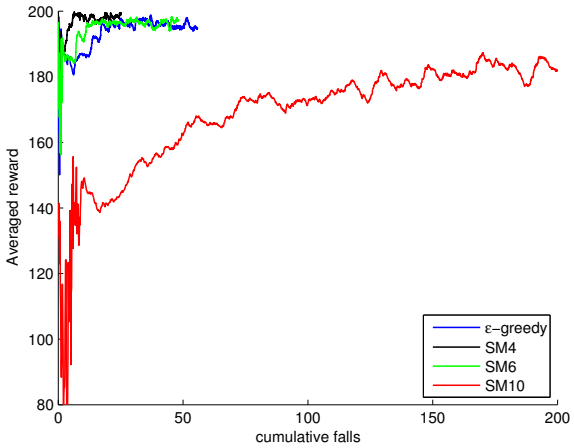
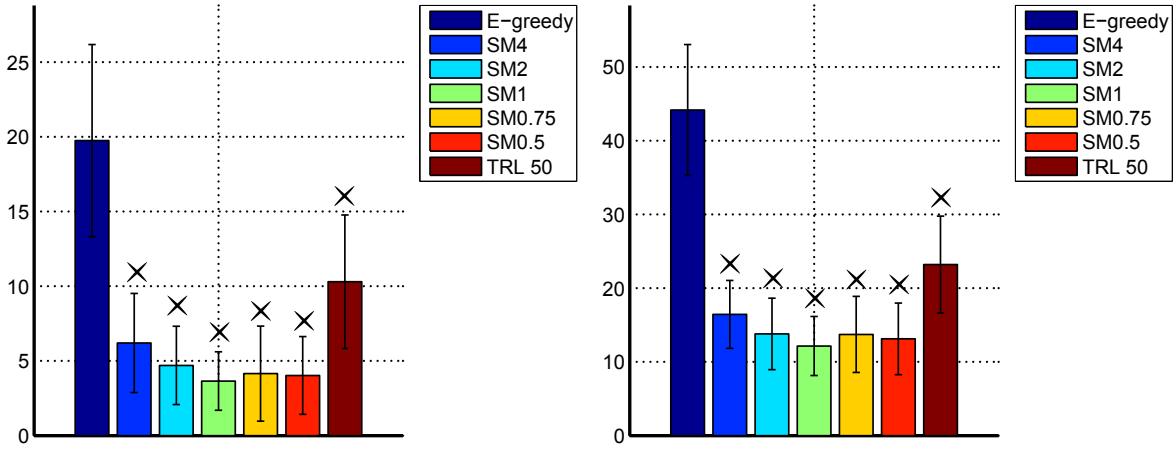


Figure 5.1: *SM with temperatures 4, 6 and 10, scenario 2. SM with a temperature of 4 performs best. The graph shows that higher temperatures perform worse.*

In this research, it is opted to hold the temperature constant, independent of the converging of the Q-function. With a dynamic temperature, it is impossible to have a proper comparison between SoftMax and TRL. Also, the main goal of a rising temperature is to optimize the exploitation of the knowledge stored in the (almost) converged Q-function. Since this research is not about optimizing

the final performance, this is of less importance. Scenario 1 is not shown here because the scenario has proven to be unfitted to show any difference in performance.

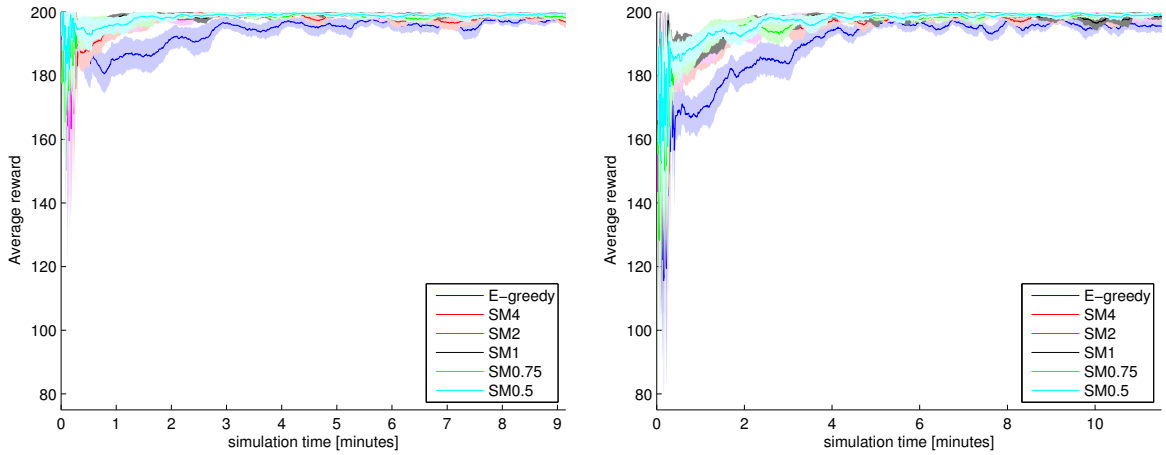
As shown by Figure 5.2a, Softmax performs clearly much better than ϵ-greedy, for all temperatures. The p-values for all temperatures are smaller than $1e^{-11}$. In the best case, with a temperature of 1, Softmax performs approximately 80% better. Figure 5.3 shows the learning curves for scenario 2 with a converged and a less converged pre-learned Q-function. Softmax learns faster than ϵ-greedy for all tested temperatures.



(a) *Falls during learning with a converged pre-learned Q-function.*

(b) *Falls during learning with a less converged pre-learned Q-function.*

Figure 5.2: *Falls during learning for Softmax and the best TRL threshold, scenario 2. Softmax performs significantly better than ϵ-greedy. For both the converged and the non-converged case Softmax performs better than the best performing TRL. Significant results are marked with an X.*



(a) *Learning curve with a converged pre-learned Q-function.*

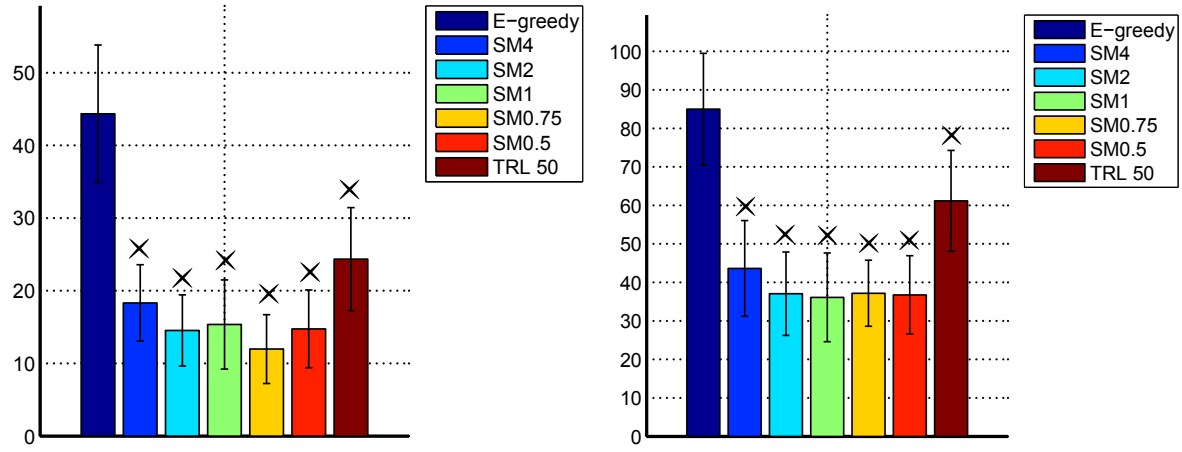
(b) *Learning curve with a less converged pre-learned Q-function.*

Figure 5.3: *Learning curves for scenario 2 with a converged and a less converged pre-learned Q-function. Softmax learns faster than ϵ-greedy for all tested temperatures.*

As can be seen in Figure 5.2b, the performance for all methods worsens again when the pre-learned Q-function is not converged. But SoftMax performs again significantly better for all tested temperature values.

Figure 5.4 shows similar results for scenario 3 as seen for scenario 2. With a converged Q-function, Softmax is able to perform approximately 70% better than ϵ-greedy. With a non-converged Q-funtion this is 55%, though the absolute result is worse than with the converged Q-function. Softmax performs better than the best TRL, for all cases. The results for scenarios 4 and 5 are similar to the results for scenarios 2 and 3.

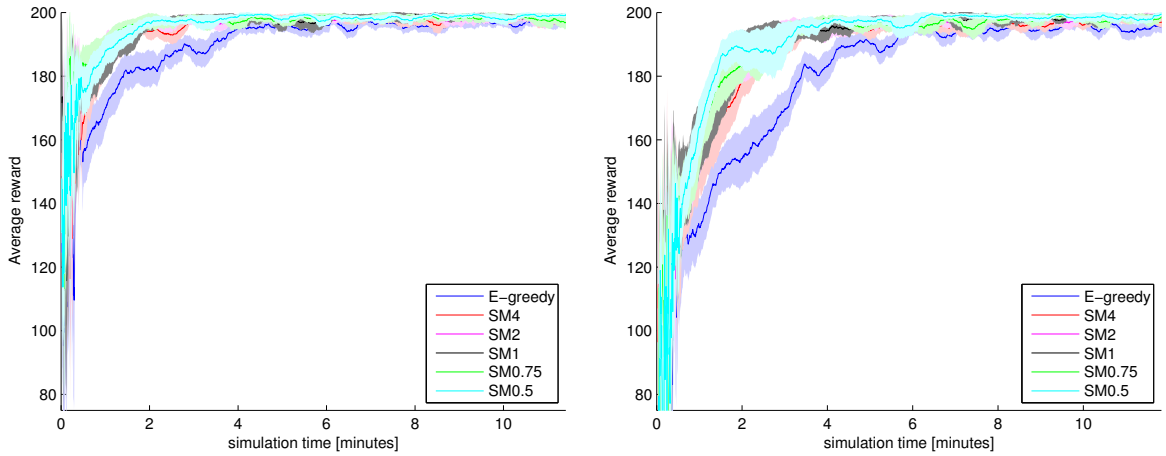
Softmax without Pre-Lreaning When Softmax is used without pre-learnin, a reduction of the amount of falls during learning of approximately 20% can be achieved, without increasing the learning time. Figure **??** shows that the results for temperatures 0.5, 1, 2 and 4 are significant, all with a p-value below $1e^{-6}$. The same τ can be used for learning with and without pre-learning.



(a) *Falls during learning with a converged pre-learned Q-function.*

(b) *Falls during learning with a non-converged pre-learned Q-function.*

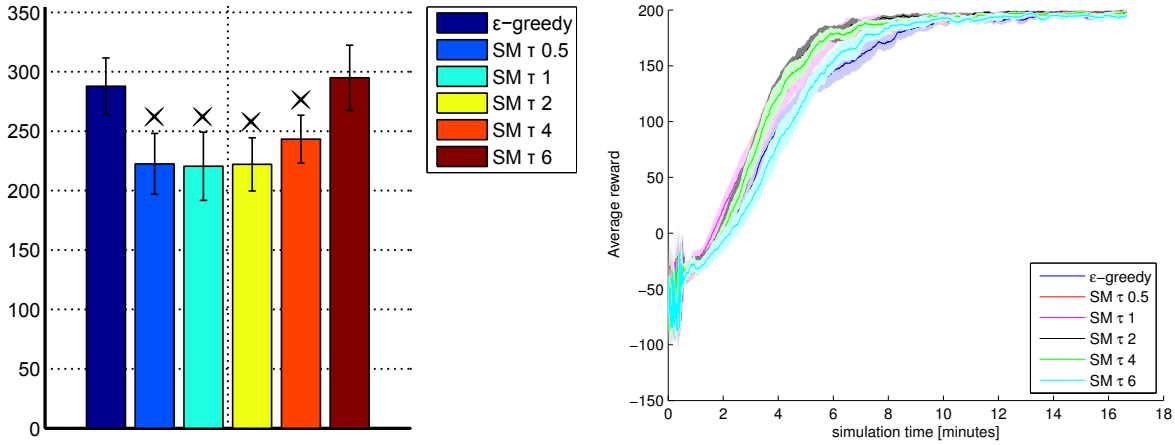
Figure 5.4: *Falls during learning for Softmax and the best TRL threshold, scenario 3. Softmax performs significantly better than ϵ-greedy. For both the converged and the non-converged case Softmax performs better than the best performing TRL. Significant results are marked with an X.*



(a) *Learning curve with a converged pre-learned Q-function.*

(b) *Learning curve with a less converged pre-learned Q-function.*

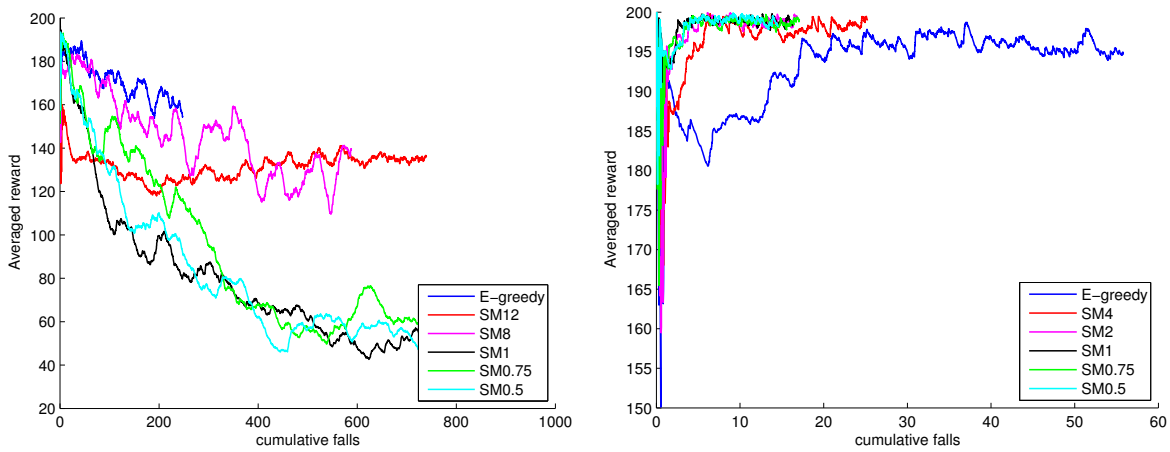
Figure 5.5: *Learning curves for scenario 3 with a converged and a less converged pre-learned Q-function. Softmax learns faster than ϵ-greedy for all tested temperatures.*



(a) *Falls during learning without pre-learning. Significant results are marked with an X.*

(b) *Learning curve without pre-learning.*

Figure 5.6: *Results for Softmax without pre-learning. Softmax performs significantly better than ϵ-greedy.*



(a) *Reward versus falls, scenario 2, with discount rate 0.96, Softmax*

(b) *Reward versus falls, scenario 2, with discount rate 0.98, Softmax*

Figure 5.7: *Reward versus falls, scenario 2, Softmax*

Discount Rate SM For Softmax action selection, the absolute change in the Q-values, as a result of the lower discount rate, does not make a difference because the selection probability of an action is dependent on it's relative Q-value. The bad performance shown in Figure 5.7a is due to the fact that a discount rate of 0.96 is too low for the problem. No measurable influence of the discount rate on the best temperature was found.

5.1 Conclusion Softmax Action Selection

It is concluded that Softmax action selections performs better than ϵ-greedy and TRL. It can reduce the amount of falls during learning with approximately 80%, compared to ϵ-greedy. Next to that, for the tested scenarios, the difference between the performance with different temperatures τ is small. This makes it easier to choose good parameters than with TRL. Using Softmax did not increase the learning time, compared to ϵ-greedy. Without pre-learning, Softmax can still reduce the amount of falls during learning with approximately 20%. The same τ was used for learning with and without pre-learning.

Chapter 6

Sarsa$_{2Q}$

In order to minimize the amount of failure during learning, a new method is introduced, named Sarsa$_{2Q}$. This method differs from Sarsa with respect to the storage of the Q-values: Sarsa$_{2Q}$ uses two Q-spaces. One Q-space has less generalization than the other. The Q-space with little generalization is used to store the expected total positive reward, the Q-space with more generalization is used to store the expected total negative reward. Sarsa$_{2Q}$ learns to avoid the failure states for the inverted pendulum problem faster than Sarsa. Choosing the same amount of generalization for the positive Q-space as for the Q-space of Sarsa is a good directive. There is no general guideline for finding a good generalization for the negative Q-space.

6.1 Working Principle

In a system that has positive and negative rewards, an agent needs to learn two things; to avoid failure and to reach the goal(s). In the case that failure occurs in (a) specific area(s), it might be advantageous to have more generalization in the state-action space representation, so the agent learns to avoid failure states quickly.
To explain the working of Sarsa$_{2Q}$ it is important to understand the effect of the generalization of the Q-space on the learning process. For an explanation of generalization, see Section 2.3 of Chapter 2.

6.1.1 The Effect of Generalization on Learning

Using a large generalization smoothens the Q-function because Q-values in a large area influence each other. As a result, the agent may learn to avoid areas, based on the estimated Q-value of only a few state-action pairs in that area. In Figure 6.1, two value functions with different generalization are shown for a grid world problem. The grey area depicts states with a low expected total reward. An agent, considering to take action 1, is not 'warned' by the fine value function, but is by the coarse value function. But smoothening obstructs the Q-function of representing sharp changes in the Q-values and might therefore deteriorate the end performance when the goal is narrowly defined. This means there is a trade-off between learning speed and end performance. And if the failure states are distributed evenly over the whole state-action space, a large generalization will not stimulate the avoidance of failure areas, because all areas have failure states.

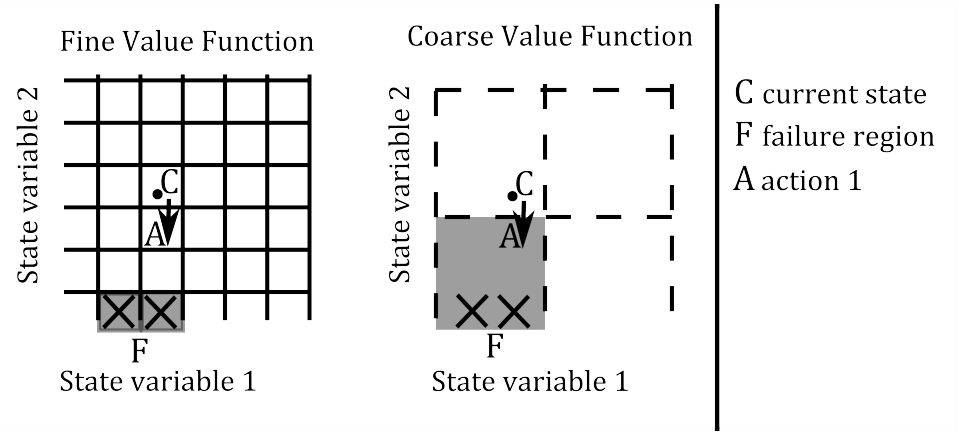


Figure 6.1: *Two Q-spaces with different generalization. The gray area of the Q-space has a low Q-value. The coarse value function can 'warn' the learner for failure state up ahead, the fine value function can not.*

6.1.2 Splitting up the Q-Space

It would be favorable to use a large generalization for representation of the negative expected rewards, so the agent learns to avoid failure states quickly. But for the storage of the positive expected rewards, less generalization might be better, so there can be a sharp change between the Q-values for the goal and all other state-action pairs. This gives rise to the idea of splitting the Q-space. One Q-space, Q^-, will estimate the expected total negative reward and shall have a coarse representation. Q^- will be updated with the negative rewards. The expected result is that the agent will learn to avoid these negative rewards very quickly. The other Q-space, Q^+, will estimate the expected total positive reward and have a finer representation. Q^+ will be updated with the positive rewards. This Q-space takes more time to learn but can use the fine representation to reach the goal. As will be discussed in the Section Future Work 6.5, the way the reward structure is split up depends on the objective. The distinction is made between the rewards that need a fine representation and the ones that do not. For this inverted pendulum case, this is equivalent to splitting up by sign.

This method, using the Sarsa update rule and using two Q-spaces, is called Sarsa$_{2Q}$. The only difference between Sarsa and Sarsa$_{2Q}$ is the difference in representation between $Q(s_t, a_t)$, $Q^+(s_t, a_t)$ and $Q^-(s_t, a_t)$. When $Q^+(s_t, a_t)$ and $Q^-(s_t, a_t)$ have the same generalization as the Q-space for Sarsa, there is no significant difference in performance, as can be seen in Figure 6.2. The non-significant difference between the graphs is due to the fact that the Q-functions are initialized with different sets of random values. The theoretical proof is given below.

The reward given at time step k can be split in a positive and a negative part:

$$r_k = r_k^+ + r_k^- \tag{6.1}$$

When Equation 6.1 is substituted in the equation for $Q(s_t, a_t)$ (Equation 2.4), $Q^-(s_t, a_t)$ and

$Q^+(s_t, a_t)$ are defined:

$$\begin{aligned} Q(s_t, a_t) &= \mathsf{E}\{\sum_{k=0}^{\infty} \gamma^k r_k\} \\ &= \mathsf{E}\{\sum_{k=0}^{\infty} \gamma^k r_k^- + \gamma^k r_k^+\} \\ &= \mathsf{E}\{\sum_{k=0}^{\infty} \gamma^k r_k^-\} + \mathsf{E}\{\sum_{k=0}^{\infty} \gamma^k r_k^+\} \\ &= Q^-(s_t, a_t) + Q^+(s_t, a_t) \end{aligned} \tag{6.2}$$

Here γ is the discount rate. Q^+ and Q^- can be updated using the same techniques as is used for updating Q, in this case Sarsa.

$$\begin{aligned} Q^+(s_t, a_t) &\leftarrow Q^+(s_t, a_t) + \alpha[r_{t+1}^+ + \gamma Q^+(s_{t+1}, a_{t+1}) - Q^+(s_t, a_t)] \\ Q^-(s_t, a_t) &\leftarrow Q^-(s_t, a_t) + \alpha[r_{t+1}^- + \gamma Q^-(s_{t+1}, a_{t+1}) - Q^-(s_t, a_t)] \end{aligned} \tag{6.3}$$

Here α is the learning rate. When the generalization is attained by using linear function approximation, the combined Q-value of a state-action pair is calculated by:

$$Q_t(s_t, a_t) = \sum_{i=1}^{n_1} \theta_t^+(i)\phi_{s,a}^+(i) + \sum_{j=1}^{n_2} \theta_t^-(j)\phi_{s,a}^-(j) \tag{6.4}$$

Here $\theta_t^+(i)$ and $\theta_t^-(j)$ are respectively the values of the tiles of Q^+ and Q^-, $\phi_{s,a}^+(i)$ and $\phi_{s,a}^-(j)$ the corresponding weights and n_1 and n_2 are the number of tiles per Q-space. The updating of $\theta_t^+(i)$ and $\theta_t^-(j)$ occurs separately. See Section 2.3 for an explanation of linear function approximation.

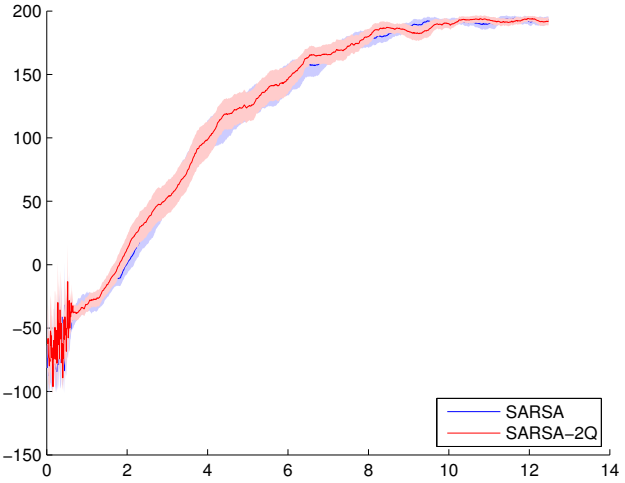


Figure 6.2: *Empirical proof Sarsa$_{2Q}$. Learning curve for Sarsa and Sarsa$_{2Q}$ with the same resolution for the Q-space of Sarsa and the Q^- and Q^+ for Sarsa$_{2Q}$. The 95% confidence interval is plotted as the bound. As can be seen, there is no significant difference between the two results.*

6.1.3 Related Work

The method for combining the two Q-values used for Sarsa$_{2Q}$ is the same as described by Sprague and Ballard [8], though implemented for different reasons. They present a new algorithm, GM-Sarsa(O), with the goal of finding approximate solutions to multiple-goal reinforcement learning problems. Instead of searching for optimal policies for the component Markov Decision Processes (MDPs) in isolation, their approach finds good policies in the context of the composite task. Different sub-goals are modeled as MDPs and share one action space. They show that combining the greatest mass (GM) arbiter action selection [9] with Sarsa results in a good solution. GM takes the sum of the Q-values over all sub-modules and choses the action with the largest sum. GM combined with Q-learning had already been used, but the writers state that the assumption that future action shall be chosen optimally is not valid when using GM. Hence, they substituted Q-learning with Sarsa. Though Sarsa and GM are used in Sarsa$_{2Q}$, the goal of the method is completely different. In this research, the two tasks modeled as MDPs are not contradicting.

Research that is more similar to this thesis is done by Grześ and Kudenko [3]. The construction of the sub-goals used here is similar to the sort of reward shaping used in their research. Their goal is to speed up convergence. They split one MDP in two; the original MDP with a state-action value function and a second MDP with a state value function. The amount of states of the second MDP is reduced by aggregating clusters of states of the first MDP that are likely to have the same optimal action. The second MDP is used for potential-based reward shaping [10]. With reward shaping implemented, the formula for Sarsa for the main MDP becomes:

$$Q_t(s,a) \leftarrow Q_t(s,a) + \alpha\{r + F(s,a,s') + \gamma Q(s',a') - Q(s,a)\} \tag{6.5}$$

with learning rate α, reward r, discount rate γ and shaping function $F(s,a,s')$. s and a are the current state and action, s' and a' are the new state and action. The reward shaping function is built as follows:

$$F(s,a,s') = \gamma_v V(z') - V(z) \tag{6.6}$$

where γ_v is the discount rate for the second MDP, z is the state of the second MDP (an aggregation of s) and V is the state-value function for the second MDP and is updated with TD(0). The value function of the second MDP is used to determine the extra reward granted for the state transition from s to s'. Though the goal is different, the method described here is very similar to the research discussed in this report.
In this thesis, splitting the reward in R^+ and R^- is necessary because the reward for balancing is narrowly defined. When the reward structure of the second MDP consist only of the R^- rewards and the value function is a state-action value function, the effect of using $F(s,a,s')$ might be similar to Sarsa$_{2Q}$. This is matter for further research.

6.2 Implementation

The inverted pendulum is an example of a system with a large state space area with failure states and a narrowly definition of the goal. The function approximator used in the inverted pendulum problem is tile coding. With tile coding, the resolution and generalization are linked, as will be explained here. An effective resolution is determined by the size of the tiles and the number of tilings. The actual resolution is the same as the generalization, see Figure 6.3a. One tiling is shown in red. The actual and effective resolution of the state space is 5. The generalization over the variables is determined by the size of the tiles. All Q-values of state-action pairs within one tile have influence on each other. In Figure 6.3b, three stacked tilings are shown. The effective

resolution of the state space is now 15 because that is the amount of different values the system can store per unit. But the generalization is still 5 and so is the real resolution. With an effective resolution, not all Q-functions can be represented because Q-values can not be set independently. They influence each other through the large tiles and therefore a limited representational power is obtained, i.e., a limited amount of sharp changes can be represented. From here on, the term resolution refers to the real resolution. To enlarge the generalization, the tiles are enlarged, which also lowers the effective resolution. To increase the effective resolution, the number of tilings can be increased, which has no effect on the generalization. Mathematically this is:

$$\begin{aligned} &\text{generalization} \sim \text{tile size} \\ &\text{effective resolution} \sim \frac{\#\text{ tilings}}{\text{tile size}} \end{aligned} \tag{6.7}$$

In this research, the real resolution and generalization are varied by changing the tile size and the effective resolution changes with it. Therefore, a subsequent test is done to determine if there is an effective resolution closer to the optimum.

Height of the Negative Reward For all tests in this section, the height of the negative reward was altered from -100 to -400. A negative reward of -100 gave less defined results. Magnifying the negative reward is the same as giving Q^- more weight.

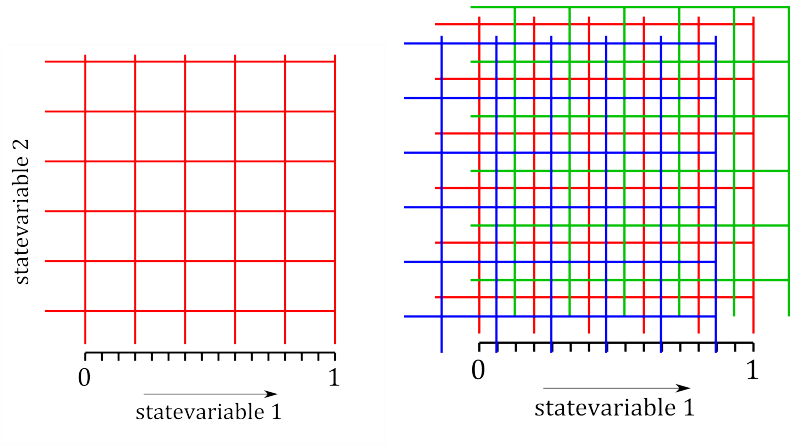
6.2.1 Optimal Resolution for Sarsa on the Inverted Pendulum

To make the comparison between Sarsa and Sarsa$_{2Q}$ fair, it is important to determine what resolution results in a reasonable balance between learning speed and end performance for Sarsa on the inverted pendulum. Since there is no set of rules to determine the optimal resolution, it has to be found empirically. As mentioned in chapter 3, the state of the inverted pendulum consists of the angle of the pole, the angle rate of the pole and the speed of the cart. Until this point, all tests were done with the following generalization and effective resolution:

	pole angle α	pole angle rate $\dot{\alpha}$	speed of the cart's wheels $\dot{\beta}$
generalization	12 tiles per 2π rad	6 tiles per $2\pi\ rad/s$	6 tiles per rad/s
effective resolution	192 per 2π rad	96 per $2\pi\ rad/s$	96 per rad/s
# of tilings	16		

The effects of the generalization and the resolution of the three state variables, $\begin{Bmatrix} \alpha \\ \dot{\alpha} \\ \dot{\beta} \end{Bmatrix}$,are not independent. But the best resolution will be sought after by assuming it is. First, a good resolution for the angle was sought after because this was found to be most critical. The resolution for the angle rate and the speed is held at the original values. Using the best resolution for the angle, results for different resolutions for the angle rate are generated. And using the best values for the angle and the angle rate resolutions, the results for different resolutions for the cart's speed is generated. In Figure 6.4, two measures of performance are plotted for the resolutions of the three state variables. The continuous line represents the falls during learning. A low value means that the learner was done learning after a small amount of falls. The dotted line represents the average amount of falls per episode at the end of the simulation, as a measure of the end performance. All

values are an average of the result of 40 lives. The simulation for the angle rate had 80 lives. For all variables holds that a higher resolution results in a better end performance. But for the speed and the angle rate the amount of falls during learning increases with an increasing resolution. The best found resolution for Sarsa was 29 tiles per 2π, 10 tiles per $2\pi\ rad/s$, 5 tiles per m/s. During the continuation of this research this resolution shall be denoted by Q_{opt}.



(a) *1 tiling* (b) *3 tilings*

Figure 6.3: *The amount of tiles determines the amount of generalization. The effective resolution is the amount of tiles times the amount of tilings.*

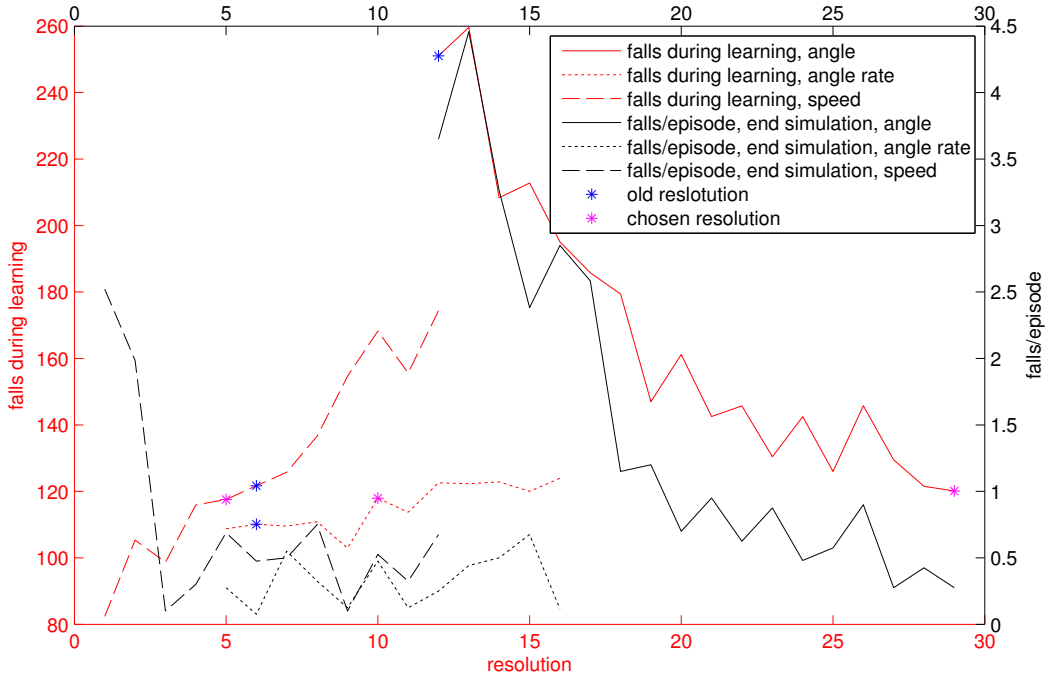
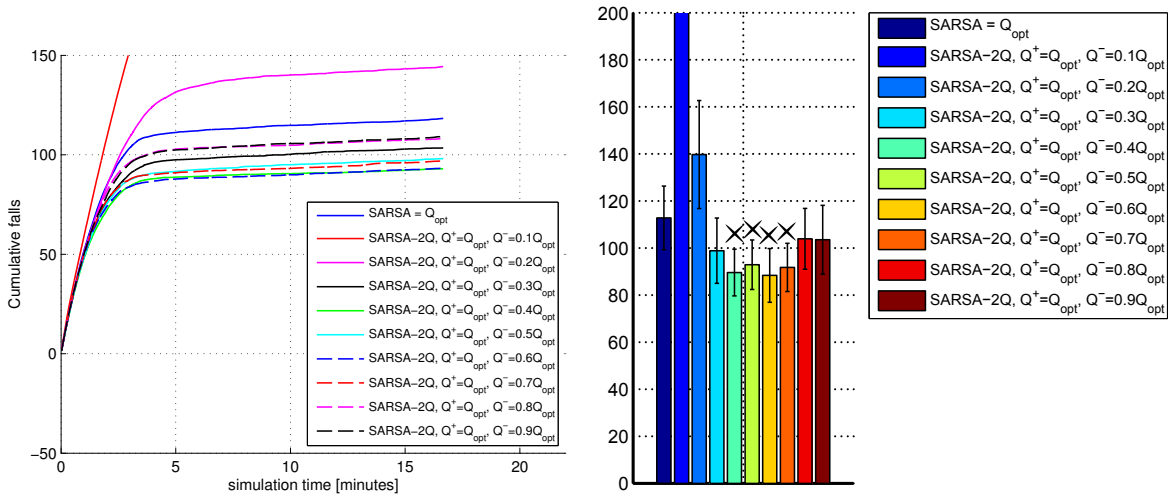


Figure 6.4: *Falls during learning and the average amount of falls per episode at the end of the simulation for Sarsa with different resolutions for the angle, angle rate and the speed. The three parameters are varied individually and started at* $[12, 6, 6]$. *The simulation for the angle rate had 80 lives, therefore the average amount of falls per episode is not exactly the same as for the chosen resolution on the angle curve.*

6.3 Results

As stated before, there are no rules to determine the right resolution for a Q-space. For Sarsa$_{2Q}$, two resolutions need to be estimated. Finding the optimal resolution for Q^-, the same way as was done for the Q_{opt} for Sarsa, was not possible. Independent optima did not add up to a combined optimum. For the results shown in Figures 6.5a and 6.5b, the resolution for Q^- is taken to be a fraction of the optimal resolution found for Sarsa and the resolution for Q^+ is taken to be the optimal resolution found in the precious section.



(a) *Cumulative falls versus time for Sarsa and Sarsa$_{2Q}$. The cumulative falls is an average over 80 lives.*

(b) *Falls during learning for Sarsa and Sarsa$_{2Q}$. Significant results are marked with an X.*

Figure 6.5: *Results for varying the resolution of Q^-, while keeping the resolution for Q^+ constant. For comparison, Sarsa with Q_{opt} is plotted. Q^- as 0.4, 0.5, 0.6 and 0.7 times Q_{opt} perform significantly better.*

The ANOVA values for the Sarsa$_{2Q}$ samples compared to the Sarsa sample are given in table 6.1. Results are significant for p-values lower than 0.01. Q^- with a resolution of 0.4, 0.5, 0.6 or 0.7 times Q_{opt} falls significantly less than Sarsa. As can be seen in Figure 6.5a, the end performance, the gradient of the cumulative amount of falls after learning, is comparable. An additional test was done with a Q^- resolution of $[\ \frac{12}{2\pi}rad \quad \frac{6}{2\pi}rad/s \quad \frac{1}{2\pi}rad/s\ ]$ which is equivalent to [0.41 0.6 0.2] times Q_{opt}. The resolution for Q^+ was the same as Q_{opt}. This test showed 25% less falls than Sarsa. Therefore, a good Q^- resolution does not necessarily have the same fractional value of Q_{opt} for each variable. A Q^+ different from Q_{opt} does not improve the result, see Figure 6.6. This is as expected because if a fine resolution is needed for standard Sarsa to represent the goal states accurately, it is also needed for Q^+. And raising the resolution might improve the end performance but slows down the learning process.

$Q^- =$	$0.1Q_{opt}$	$0.2Q_{opt}$	$0.3Q_{opt}$	$0.4Q_{opt}$	$0.5Q_{opt}$	$0.6Q_{opt}$	$0.7Q_{opt}$	$0.8Q_{opt}$	$0.9Q_{opt}$
p-value	0	1.5e-4	0.006	3.5e-7	1.5e-5	3.7e-7	4.2e-6	0.069	0.076
% less falls	-211	-23	12	21	18	22	19	8	8

Table 6.1: *Q^+ is Q_{opt}, the resolution of Q^- is a varying fraction of Q_{opt}.*

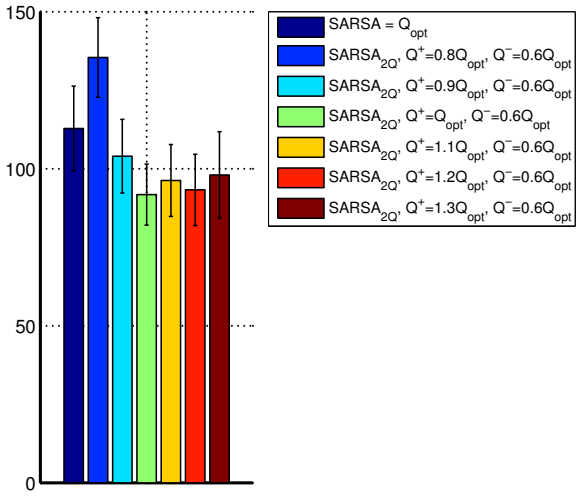


Figure 6.6: *Sarsa$_{2Q}$ with a varying resolution for Q^+, Q^- is constant at $0.6Q_{opt}$. There is no significant improvement to be gained by using a Q^+ different from Q_{opt}.*

6.3.1 No Quick Fix for Resolution of Q^-

It is unlikely that there is a fixed fraction of Q^+ for the resolution of Q^-, that performs well for all systems. A system with a Q-space that can be thought of as having a large 'bad' part and a small 'good' part, is expected to work better with a large generalization for Q^- than a system with a Q-space with 'bad' and 'good' parts mixed. In the last case, Sarsa$_{2Q}$ might not work at all. The resolution for Q^- is expected to depend on the dispersion of the failure states and the distance between the goal and the failure states. With a large dispersion, a large generalization for Q^- will downgrade the whole state-action space evenly and therefore have no effect. With a short distance between goal and failure and a large generalization, Q^- can have a low expected reward for the region of the goal.

6.3.2 Changing the Effective Resolution for Q^-

Until this point, the effective resolution of Q^- changed with the change in generalization, the number of tilings stayed unaltered. The following tests are to determine if the previously found results can be improved by altering the resolution, without altering the generalization. The number of tilings needs to be a power of 2 because this is a requirement of the algorithm used to position the tilings, **??**. In Figure 6.7, the amount of falls during learning is shown for four different amounts of tilings. The resolution for Q^- is $0.6Q_{opt}$, the resolution for Q^+ is Q_{opt}. The best result is obtained by using 16 tilings. This is the same amount as was used for all preceding tests, therefore, in this case, the results can not be improved by altering the resolution.

6.3.3 Changing the Learning Rate

With larger tiles, it is customary to lower the learning rate α. An update on a tile that consists of many states-action pairs has a higher uncertainty than an update on a tile that consists of only a few states-action pairs. Hence, a large tile requires more averaging. By using a lower learning rate, old Q-values have a larger share in the average. The question rises if changing the learning rate is

desirable for Q^-. Again, first a good learning rate for Sarsa is determined. The learning rate for all preceding tests was 0.4. Figure 6.8a shows that the chosen optimal resolution works best with a learning rate of approximately 0.7. In Figure 6.8b, the result for Sarsa$_{2Q}$ with $Q^+ = Q_{opt}, \alpha = 0.7$, $Q^- = 0.5Q_{opt}$ is shown. The learning rate for Q^- is varied. Testing with ANOVA shows that Sarsa$_{2Q}$ with an α of 0.7 for Q^- is significantly better than the optimal Sarsa (p-value = 0.0038). The fact that the optimal α is the same for both Q-spaces is expected to be unlinked. Against the expectation, in this case the learning rate for Q^- should not be lower than the learning rate for Q^+, while the resolution is significantly lower. A possible explanation for the high α of Q^- is that it is desirable that the agent learns to avoid the failure states with the least possible amount of experience. Therefore it might be better to have an unjust negative view of a state than to learn more prudently.

The best performing Sarsa$_{2Q}$, $Q^+ = Q_{opt}$, $Q^- = 0.5Q_{opt}$, $\alpha = 0.7$, falls approximately 10% less than Sarsa, see Figure 6.8. To determine if Sarsa$_{2Q}$ increases the learning time, the learning curve for Sarsa and Sarsa$_{2Q}$ is shown in Figure 6.9. It can be concluded that Sarsa$_{2Q}$ does not increase the learning time.

Comparing the result for Sarsa$_{2Q}$ to the results for Softmax action selection and Threshold Restricted Learning is flawed for two reasons. Firstly, TRL and SM were tested with a learning rate of 0.4, which turned out not to be optimal. And a fair comparison can only be made with results without pre-learning.

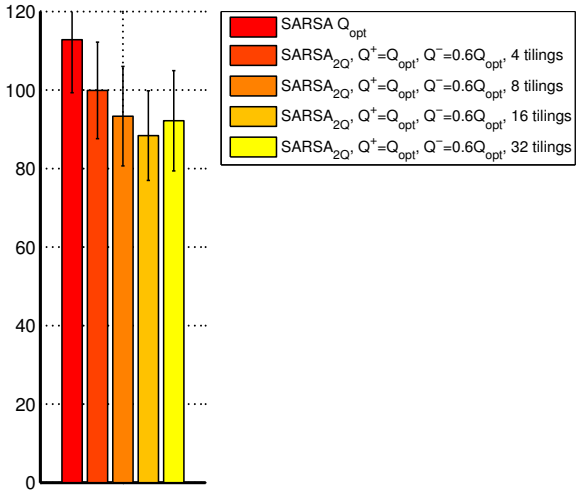
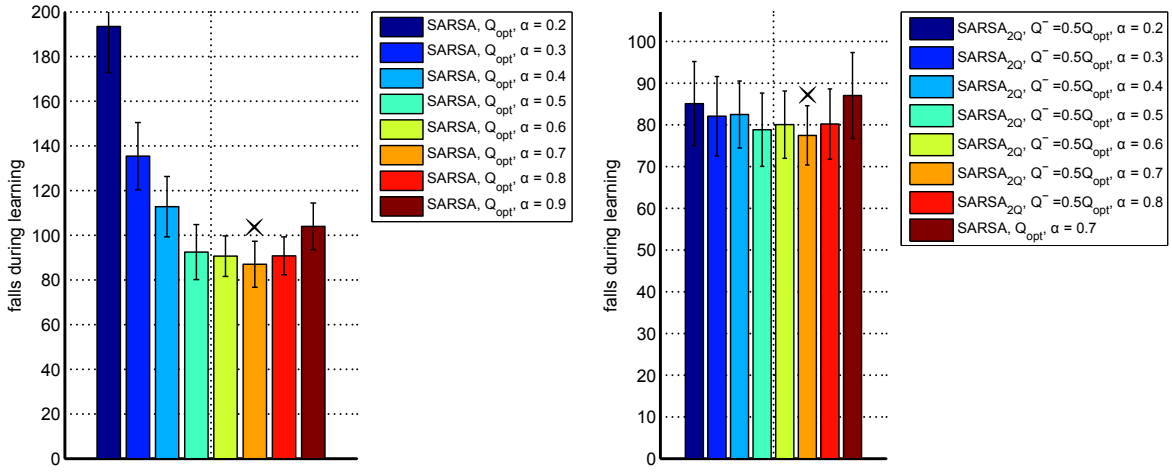


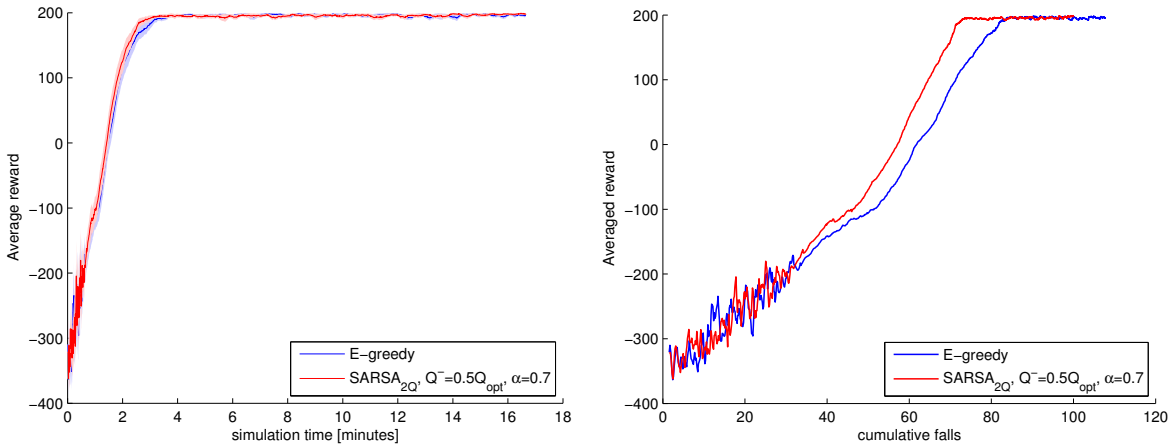
Figure 6.7: *Sarsa$_{2Q}$, the amount of falls during learning for different numbers of tilings for Q^-. All preceding test were done with 16 tilings for Q^-. Changing the amount of tilings does not improve the result.*



(a) *Falls during learning for Sarsa with different learning rates.*

(b) *Falls during learning for Sarsa$_{2Q}$ with different learning rates for Q^-. α for Q^+ is 0.7.*

Figure 6.8: *Falls during learning for Sarsa and Sarsa$_{2Q}$, with different α for Q and Q^-. The result for Sarsa can be significantly improved by changing the learning rate from 0.4 to 0.7. Sarsa$_{2Q}$ with a learning rate of 0.7 for the Q^- gives a significant better results than Sarsa, it reduces the amount of falls by approximately 10%. Significant results are marked with an X.*



(a) *Learning curve Sarsa and the best performing Sarsa$_{2Q}$.*

(b) *Average total reward versus the amount of falls for Sarsa and the best performing Sarsa$_{2Q}$.*

Figure 6.9: *The learning curve and the average total reward versus the amount of falls for Sarsa and the best performing Sarsa$_{2Q}$. It is shown that Sarsa$_{2Q}$ does not increase the learning time.*

6.4 Conclusion Sarsa$_{2Q}$

Sarsa$_{2Q}$ has proven to reduce the amount of falls during learning on the inverted pendulum model with respect to ϵ-greedy, without increasing the learning time.

A good way to determine a good resolution for Q^+ is taking the same resolution as was best for the Q-space of Sarsa. Higher or lower resolution did not give a significant better result. There is no rule found to help finding a good resolution for Q^-. The resolution for Q^- is expected to depend on the dispersion of the failure states and the distance between the goal and the failure states. For the inverted pendulum problem, the best found resolution for Q^- was 0.6 times the resolution of Q^+.

The best α for Q^- was 0.7, which was higher than expected because 0.7 was also the best learning rate found for Sarsa, while the resolution for Sarsa was 1.6 times higher than the resolution for Q^-. A possible explanation for the high α of Q^- is that it is desirable that the agent learns to avoid the failure states with the least possible amount of experience. Therefore it might be better to have an unjust negative view of a state than learn more prudently.

The Applicability of Sarsa$_{2Q}$ Sarsa$_{2Q}$ performs better than Sarsa on a system with low dispersion of failure states and a narrow definition of the goal. The resolution for Q^+ can be taken the same as the resolution used for Sarsa. But for choosing the resolution for Q^- there is no guideline, except that is should be lower than the resolution for Sarsa. A wrongly chosen resolution for the Q^- of Sarsa$_{2Q}$ can have results that are worse than the results for Sarsa. For the inverted pendulum case, the penalty for falling was changed from -100 to -400 to obtain significant results.

6.5 Future Work

The utility of Sarsa$_{2Q}$ would improve if there was a guideline to estimate a good generalization and resolution for Q^- and Q^+. It is also interesting to test Sarsa$_{2Q}$ while using a function approximator that can set the generalization and resolution independently, e.g., radial basis functions. With tile coding, only the effective resolution can be set independently from the generalization.

Sarsa$_{2Q}$ might prove useful in achieving a different objective, e.g., increasing the convergence rate. One might think of a coarse Q-space that covers the whole state space and a finer Q-space that only covers an area of states that need this precision. States covered by both Q-spaces will have a composed Q-value. This results is a reduction of the amount of states. Both Q-spaces might have the complete reward structure or might have parts of the total reward structure. In the latter case, only rewards that need to be estimated precisely shall be used in the fine Q-space.

Chapter 7

Conclusion

With the objective of reducing the amount of falls during learning, three methods were tested on a simulation of an inverted pendulum. All three tested methods avoided failure more effectively than Sarsa ϵ-greedy, without increasing the learning time.

Threshold Restricted Learning (TRL) was introduced. This method differs from ϵ-greedy action selection in the choice of the exploratory action. Where ϵ-greedy takes truly random actions, TRL takes a random action with a Q-value above a fixed threshold. When no action above the threshold is available, a random action is chosen. The method is tested in combination with pre-learning. A Q-function is learned on the pre-learn simulation and used as initialization of the Q-function of the actual simulation. The threshold can be estimated using the rule of thumb:

$$\text{Thres}_{opt} = \gamma^{n^*} * R_{goal} \tag{7.1}$$

Where $Thres_{opt}$ is the expected optimal threshold, γ the discount rate, n^* the expected average amount of transitions per *successful* episode *during learning* and R_{goal} the positive end reward. This rule of thumb estimates the Q-value of the first state-action pair of a successful episode during the learning stage.

TRL has shown to learn at the same speed as ϵ-greedy, while visiting less failure states. On a simple system, like the model of the inverted pendulum used in this research, TRL performs better than ϵ-greedy only when the difference between the pre-learn simulation and the actual simulation are too large to be realistic. Using the rule of thumb, a good estimation of the optimal threshold can be given. The largest reduction in the amount of falls achieved by TRL in this research was 50%. Without pre-learning, TRL can still reduce the amount of falls during learning by 7%.

Softmax action selection reduces the falls during learning, compared to ϵ-greedy, and performs better than TRL. Next to that, for the tested scenarios, the difference between the performance with different temperatures τ is small, which makes tuning easier than for TRL. Using Softmax did not increase the learning time, compared to ϵ-greedy. The largest reduction in the amount of falls achieved by Softmax in this research was 80%. Without pre-learning, Softmax can reduce the amount of falls during learning by approximately 20%.

Sarsa_{2Q} is a new method that uses two Q-spaces. One Q-space has less generalization than the other and is used to store the expected total positive reward. The second has more generalization and is used to store the expected total negative reward. Sarsa_{2Q} learns to avoid the failure states for the inverted pendulum problem faster than Sarsa and does not increase the learning time. Choosing the same amount of generalization for the positive Q-space as for the Q-space of Sarsa is a good directive. The best found generalization for the negative Q-space for the inverted pendulum was $\frac{1}{2}$ times the generalization of the positive Q-space. The largest reduction in the amount of

falls achieved by Sarsa_{2Q} in this research was approximately 20%. It is important to note that this is without pre-learning.

Softmax action selection outperforms Threshold Restricted Learning. Only when a system needs a large amount of exploration it can be beneficial to use TRL. In that case, with its discontinuous probability function, it can explore extensively above the threshold and still avoid state-action pairs with Q-values below the threshold. Softmax can not have a small difference in probability between two high Q-values and at the same time have a large difference in probability between a high and low Q-value. Sarsa_{2Q} performs well without a pre-learned Q-function. It does however require good insight in the system in order to estimate good resolutions for the two Q-spaces. It is promising that Sarsa_{2Q} can have a broader use than just avoiding failure states. The coarse Q-space is not necessarily updated with the negative rewards, it can be updated with all rewards that need less precision. It is possible, e.g., that the goal reward is used for the update of both Q-functions. Through the coarse Q-function the learner can find the areas with high expected rewards faster, through the fine Q-function it can find the exact location of the goal. Or by dividing states that need high precision and states that need low precision over the two Q-spaces, a reduction of the amount of states can be achieved.

A good follow up on this research would be the testing of Sarsa_{2Q} on LEO. It is expected that a larger reduction in the amount of falls during learning can be achieved on LEO than on the inverted pendulum because the inverted pendulum is a relative simple system. The best tuned Sarsa only fell 80 times, see Figure 6.8a, and this is hard to improve. Using Softmax with pre-learning on LEO is also a promising experiment because there already exists a simulation of LEO. And it might be interesting to combine Sarsa_{2Q} with Softmax. This would combine the advantage of effective Q-function representation with the cautious action selection. This combination is expected to perform better than the individual methods.

As mentioned just now, the inverted pendulum is not an optimal test simulation. The system is too simple and any method learns very fast on it. Therefore, it is difficult to achieve significant ameliorations in performance. It is expected that tests on a more complicated system will show even better results.

Bibliography

[1] Richard S. Sutton and Andrew G. Barto
Reinforcement Learning: An Introduction
1998, MIT Press

[2] Kenji Doya, Kazuyuki Samejima1, Ken-ichi Katagiri, and Mitsuo Kawato
Multiple Model-based Reinforcement Learning
November 2, 2001

[3] Marek Grześ and Daniel Kudenko
Multigrid Reinforcement Learning with Reward Shaping
2008, In: ICANN 2008, Part I, LNCS 5163, pp. 357-366

[4] Sooraj Bhat, Charles L. Isbell Jr., Michael Mateas
On the Difficulty of Modular Reinforcement Learning for Real-World Partial Programming
2006, Proceeding on the National Conference on Artifical Intelligence, vol. 21, number 1, page 318

[5] Christopher J. Hanna, Raymond J. Hickey, Darryl K. Charles, and Michaela M. Black
Modular Reinforcement Learning Architectures for Artificially Intelligent Agents in Complex Game Environments
2010, IEEE Symposium on Computational Intelligence and Games (CIG)

[6] Ono, N. and Fukumoto, K.
A Modular Approach to Multi-agent Reinforcement Learning
1997, Distributed Artificial Intelligence Meets Machine Learning Learning in Multi-Agent Environments, pages 25-39

[7] Andrew Y. Ng, Daishi Harada, Stuart Russell
Policy invariance under reward transformation: Theory and application of reward shaping
1999, Proceedings of the Sixteenth International Conference on Machine Learning, pages 278-287

[8] Nathan Sprague, Dana Ballard
Multiple-Goal Reinforcement Learning with Modular Sarsa(O)
2003, In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Workshop paper.

[9] J. Karlsson.
Learning to Solve Multiple Goals
1997, PhD thesis, University of Rochester

[10] A.Y. Ng, D. Harada, S. Russell
Policy Invariance under Reward Transformation: Theory and Application to Reward Shaping
1999, In: Proceedings of the 16th International Conference on Machine Learning, pages 278-287

[11] Fujiwara, K. and Kanehiro, F. and Kajita, S. and Kaneko, K. and Yokoi, K. and Hirukawa, H.
UKEMI: Falling Motion Control to Minimize Damage to Biped Humanoid Robot
2002, Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2521-2526

[12] Bridle, J.S.
Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition.
1990, In: F.Fogleman Soulie and J.Herault (eds.), Neurocomputing: Algorithms, Architectures and Applications, Berlin: Springer-Verlag, pp. 227-236

[13] Erik Schuitema, Martijn Wisse, Thijs Ramakers, and Pieter Jonker.
The design of LEO: a 2D bipedal walking robot for online autonomous reinforcement learning.
2010, in IEEE/RSJ International Conference on Intelligent Robots and Systems

[14] Peters, J. and Schaal, S.
Policy Gradient Methods for Robotics
2006, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2219-2225

[15] Sutton, R.S.
Dyna, an integrated architecture for learning, planning, and reacting.
1991, in ACM SIGART Bulletin, vol. 2, number 2, pages 160-163

[16] Dietterich, T.G.
Hierarchical reinforcement learning with the MAXQ value function decomposition.
1999, in Arxiv preprint cs/9905014

[17] Miller, WT and An, E. and Glanz, F. and Carter, M.
The design of cmac neural networks for control.
1990, in Adaptive and Learning Systems, volume 1, pages 140-145

[18] Douglas Adams
The Hitchhiker's Guide to the Galaxy
1989, ISBN: 0517542099 / 0-517-54209-9