Learning Parameter Selection in Continuous Reinforcement Learning

Attempting to Reduce Tuning Effords

J.C. van Rooijen

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Learning Parameter Selection in Continuous Reinforcement Learning

Attempting to Reduce Tuning Effords

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

J.C. van Rooijen

July 11, 2012

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

The reinforcement learning (RL) framework enables to construct controllers that try to find find an optimal control strategy in an unknown environment by trial-and-error. After selecting a control action, the controller receives a numerical reward. The reward signal is based on the current state of the environment and the applied control action. The controller aims to maximize the cumulative reward, known as the return. In this thesis actor-critic and critic-only RL algorithms are considered. Actor-critic algorithms consist of an element that selects the actions (the actor) and an element that learns the expectation of the return (the critic). This expectation is captured in a value function. The critic is used to improve the control policy of the actor. Critic-only algorithms select the action by direct optimization over a value function.

Before a RL algorithm can be applied to a control problem, a number of learning parameters need to be set. The optimal values of some of these parameters are highly problem dependent. It is not straight forward how these parameters should be chosen and often these are often determined by trying a large set of parameters.

The main focus of thesis is to devise an action selection method that is able to select continuous actions without problem dependent parameters. Two approaches are taken: first, it is investigated if Levenberg Marquardt (LM), a popular optimization method, can be used to determine the actor update step. Second, an action selection method is treated that lacks an explicit actor, called Value-Gradient Based Policy (VGBP).

The LM algorithm uses the gradient and the Hessian to compute the update step. Therefore the policy gradient and Hessian need to be found. A novel actor-critic method has been devised, called Vanilla Actor-Critic (VAC), that efficiently learns the policy gradient. On the inverted pendulum swing-up task this algorithm outperformed Natural Actor-Critic (NAC). A number of different approaches have been taken to approximate the policy Hessian, but none delivered a proper Hessian estimate. Therefore, no LM actor update method was created.

In VGBP the action is found by optimization of the right hand side of the Bellman equation. VGBP uses the provided reward function and a process model for this optimization. The process model is learned online using local linear regression (LLR). Due to the efficient use of information VGBP shows fast learning on the pendulum and a 2-DOF robotic arm.

Table of Contents

Acknowledgements

I am grateful to all who have helped with the successful completion of this thesis: my supervisors, colleagues , friends and family.

First of all, I would like to thank my supervisors who offered me this very challenging project. Ivo Grondman for the enjoyable weekly progress meetings in which you provided me useful knowledge, new insights and motivation, but also for thinking along about the faced problems and of for correcting my writing. Robert Babuška for making me see the bigger picture and for always challenging all the gained insights I took for granted. I learned a lot from the discussions during the progress meetings and from the useful feedback you provided me on my work.

I am grateful to my father, mother and brother for being so supportive, although going to tough time themselves. And of course I am thankful to Marlous, who had to endure the moments that I was pondering about my thesis at night and who I had to tell all my new findings and little frustrations.

Finally I want to thank all my friends. My roommates for their support, mentally and by the use of your computers for simulations and experiments. All my colleagues with whom I spent a year in the "master kelder". A special thanks to Lex, Olivier, Sander and Martijn for reading my work.

Thank you all, I could not have done this without you!

Delft, University of Technology J.C. van Rooijen
July 11, 2012

Chapter 1

Introduction

Through a rapid development in telecommunication and computer technology, components for robots have become widely available for affordable prices. It becomes profitable to use robots for an increasing range of tasks. These tasks involve increasingly complex systems in more complex environments. In many cases classical control methods cannot be applied to these control problems, because:

- The environment is unknown or unpredictable. This can be caused by contact and friction dynamics which cannot be modeled properly, or man-machine interaction.
- The objective is clear, but the solution is not. For instance in a game of chess the objective of winning the game is clear, but finding the winning strategy is computationally intractable.

For these scenarios it is useful to have a controller which finds a solution to the control problem by itself. Reinforcement learning (RL) is a framework inspired by human and animal learning that enables an agent to become proficient in a task in an unknown environment by trial-and-error (Lewis and Vrabie, 2009). The agent receives feedback on its behavior via a reward signal. A very popular branch of RL algorithms are actor-critic methods. Actor-critic methods consist of an element that determines the behavior (the actor) and an element that evaluates the performance (the critic). The feedback of the critic is used to improve the control policy of the actor. The learning speed and the quality of the learned policy are determined by the update rule of both the actor and critic.
For the critic numerous methods have been devised to optimize the update rule. Advanced update schemes are enhanced gradient ascent (Dabney and Barto, 2012; Geist and Pietquin, 2010; Ni et al., 2011; Powell, 2007) and least-squares methods (Boyan, 2002). In general the actor is updated with a gradient ascent update rule. Often used parameters in this ascent update are the learning rate or learning rate schedule, and the momentum term. The choices made in the parameters of the update rule have a large influence on the learning behavior of the agent. The optimal parameters are often highly problem dependent. To the author's best knowledge, no proper methods have been devised to set the parameters in the actor update.

1-1 Research Objectives

The goal of this thesis is to devise a method that removes the dependency of the action selection on problem dependent learning parameters. This method has to be applicable to different problems without the need to tune any parameters for the specific setup. Therefore the learning behavior of the algorithm is verified and compared to other algorithms on the inverted pendulum swing-up task as well as a robotic manipulator. Two different approaches are taken: first, ideas from classical optimization are used to improve the actor update. Second, an action selection method is discussed that is a direct optimization over the expected return of actions rather than a mapping from states to actions.

1-1-1 Automatic Learning Rate Selection

Gradient descent methods confine a large branch of classical optimization. Many methods have been devised to choose an efficient descent step (Luenberger and Ye, 2008). The Levenberg Marquardt (LM) algorithm is among the most popular nonlinear optimization methods. LM uses first and second order gradient information to determine the size and the direction of the descent step. In learning, LM has been successfully applied to neural networks. Numerous successful utilizations are found in supervised learning (Zapateiro and Luo, 2007) and more recently also in actor-critic RL in the critic update (Ni et al., 2011). LM is often used in neural networks because of its good convergence properties (Hagan and Menhaj, 1994). The first part of this project is dedicated to the implementation of LM in the actor update. LM requires the gradient and the Hessian. A novel actor-critic method has been devised that learns the policy gradient, called Vanilla Actor-Critic (VAC). A number of methods to acquire the policy Hessian have been applied to actor-critic LR. None of these methods succeeded to estimate the Hessian quickly enough to devise a proper LM actor update rule.

1-1-2 Removing the Actor

The goal of the actor is to find the action that has the highest expected return. In actor-critic methods a parameterized controller is used for this task. For this actor choices have to be made for the initial parameters in both the controller and the update rule. The need for initial parameters is inherently connected to this action selection method. If the results of all actions are known, the action can be selected by finding the action which yields the highest return directly. In the second part of this thesis the Value-Gradient Based Policy (VGBP) algorithm is treated (Doya, 2000). It learns a model of the environment and uses the acquired insight to select its actions. Because the action is selected with a simple optimization technique, the actor, including its initial parameters, is obsolete. In this thesis work local linear regression (LLR) is applied to VGBP, because this approximator has shown to learn a proper process model quickly (Grondman et al., 2012a). Furthermore, a learning rate-free analogue of VGBP is derived and tested on the inverted pendulum swing-up task and a regulator task of a robotic arm.

1-2 Outline of the Thesis

The RL framework is introduced in Chapter 2. The possibility to implement LM in RL is explored in Chapter 3 to 5. In Chapter 3 LM is introduced. The next two chapters are devoted to VGBP. This method is introduced in the form of a paper in Chapter 6. A more comprehensive discussion on the test results is given in Chapter 7. A comparison of the algorithms devised in both approaches is given in Chapter 8. The thesis is concluded with a summary of the main findings together with recommendations for future work in Chapter 9.

Chapter 2

Reinforcement Learning

This chapter introduces the reinforcement learning (RL) framework. This framework allows an agent to learn an optimal control strategy. First, the main components of the control problem will be discussed. Then, the Markov decision process will be treated. A number of algorithms that solve the RL problem are introduced.

2-1 The Reinforcement Learning Problem

Reinforcement learning is a class of algorithms that enables an *agent* to solve problems by interaction with an environment for an extended period of time. The goal of the reinforcement learning problem is not known to the agent, but is represented to the agent by a reward signal, which is a measure of the quality of the choices made by the agent. The agent tries to optimize its behavior to maximize the accumulated reward, known as the return. A schematic overview of the problem is given in Figure 2-1. The *agent* is defined as the decision maker (Babuška, 2010). The inputs to the agent are:

- A state $s \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states, also known as the state space.
- A reinforcement signal $r \in \mathbb{R}$ in the set of reinforcement signals.

The output of the agent is an action $a \in \mathcal{A}$, where $\mathcal{A}$ is the set of actions. The mapping from states to actions is the *policy* π. For deterministic policies this is $\pi : \mathcal{S} \to \mathcal{A}$. Then $\pi(s)$ indicates the action selected in state s under policy π. For a stochastic policy the mapping is $\mathcal{S} \to \Omega(\mathcal{A})$. $\Omega(\mathcal{A})$ is the set of all probability distributions over $\mathcal{A}$. With $\pi(s, a)$ the probability of taking action a in state s is indicated. For discrete and continuous action spaces respectively, the stochastic policy has to satisfy:

$$\sum_{a \in \mathcal{A}} \pi(s, a) = 1, \qquad \int_{a \in \mathcal{A}} \pi(s, a) = 1 \qquad \forall s. \tag{2-1}$$

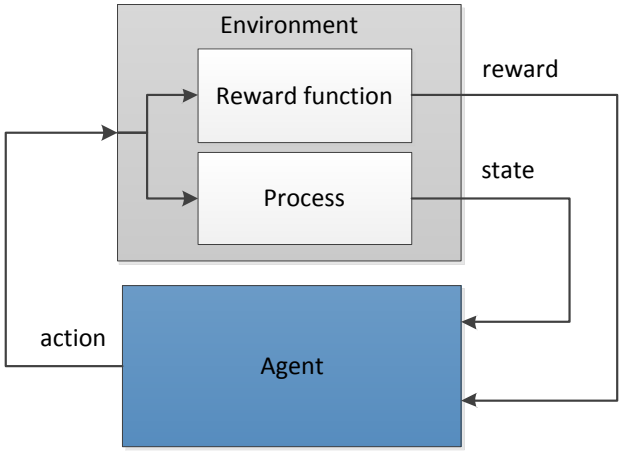


Figure 2-1: Overview of the RL-problem

Because most RL algorithms operate in discrete time, the optimization of the reward is solved as a sequential decision making problem. The state and action taken at time instant k are then denoted by s_k and a_k.
The *environment* is everything outside the decision maker. Unlike in Artificial Intelligence (AI) (Russell and Norvig, 2003), sensors and actuators are considered to be part of the environment. The environment can be characterized along several dimensions:

- Fully observable or partially observable. The (sensory) input can give the agent a complete representation of the current state, or only a partial view. It is effectively fully observable if all the relevant information in the state is obtained by the agent.
- Deterministic or stochastic. When the current state and action completely determine the next state, the environment is deterministic, but if the transitions involve probabilities it is stochastic.
- Stationary or non-stationary. The environment is stationary if the state transition probabilities and the corresponding reward do not depend on time.
- Discrete or continuous. If the state can take any value (in a certain interval), the state space is continuous, otherwise it is discrete. The same holds for the action space.

2-1-1 Markov Decision Processes

The model behind the RL problem is the Markov decision process (MDP). The MDP is the 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. The evolvement of the state through the actions of the agent is captured in a *transition model*. The transition from state s to the next state s' under action a is indicated as $s \overset{a}{\longmapsto} s'$. It is assumed that the transition model satisfies the Markov property:

the transition to the current state depends only on the previous state and the action taken. More generally, s contains all relevant information to determine the distribution over s':

$$\begin{aligned} &P(s_{k+1} = s', r_{k+1} = r \mid s_k, a_k) \\ &= P(s_{k+1} = s', r_{k+1} = r \mid s_k, a_k, r_k, s_{k-1}, a_{k-1}, \ldots, r_1, s_0, a_0) \qquad \forall s' \in \mathcal{S}\ ,\ \forall r. \end{aligned} \tag{2-2}$$

The probability that this transition is made is determined by the transition model, which for discrete state and action spaces is defined by the mapping $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. Then,

$$\mathcal{P}(s, a, s') = P(s_{k+1} = s' \mid s_k = s, a_k = a). \tag{2-3}$$

For continuous spaces the mapping $\mathcal{P}$ is a state transition probability density function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$. Because the state space is continuous, the transition function describes the chance of reaching a region. The probability of reaching state s_{k+1} in the region $\mathcal{S}_{k+1} \subset \mathcal{S}$ by applying action a_k is:

$$P(s_{k+1} \in \mathcal{S}_{k+1} \mid s_k, a_k) = \int_{\mathcal{S}_{k+1}} \mathcal{P}(s_k, a_k, s')\mathrm{d}s'. \tag{2-4}$$

The reward r is determined by the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. The reward function $\mathcal{R}(s, a, s')$ determines the reward for the transition $s \xmapsto{a} s'$. The reward function has to be constructed with care, because a wrongly designed reward functions can make the agent pursue other goals than those intended by the designer (Batalov; Kober and Peters, 2012).

2-1-2 Value Functions

To determine the performance of the agent in the long run, the rewards received in each step have to be accumulated and weighted to obtain the expected return. The expected return while following policy π, weighted by some function f is given by:

$$\rho(\pi) = E\left\{f(r_1, r_2, \ldots) \mid s_0, \pi\right\}. \tag{2-5}$$

There are multiple types of returns, depending on how the future is taken into account. In this thesis the infinite-horizon average return and the infinite-horizon discounted return are used. The *average return* is the expectation of the sum of the average reward received when following policy π for an infinite amount of time:

$$\rho(\pi) = \lim_{K \to \infty} E\left\{\frac{1}{K+1} \sum_{k=0}^{K} r_{k+1}\right\}. \tag{2-6}$$

For continuous state and action spaces this can be rewritten as:

$$\rho(\pi) = \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \pi(s, a) \int_{\mathcal{S}} \mathcal{P}(s, a, s')\mathcal{R}(s, a, s')\ \mathrm{d}s'\ \mathrm{d}a\ \mathrm{d}s, \tag{2-7}$$

where $d^{\pi}(s) = \lim_{k \to \infty} P(s_k = s \mid s_0, \pi)$ is the steady state distribution, which denotes the probability of being in state s while following policy π. This return model makes no distinction between receiving a high reward immediately or in the long run. The *infinite-horizon discounted return* however, makes this distinction and is denoted by:

$$\rho(\pi) = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} \;\middle|\; s_0, \pi\right\}, \tag{2-8}$$

where the parameter γ is the *discount factor*, with $0 \leq \gamma < 1$. The discount factor can be seen as a measure for the farsightedness of the agent. It is a balance between favoring actions that have a high immediate reward and actions with a possible higher future return. Furthermore it ensures that the expected return is bounded for bounded rewards. For continuous state and action spaces, the discounted return is also calculated with Equation (2-7), where, instead of the stationary steady state distribution, the discounted steady state distribution $d^\pi(s) = \sum_{k=0}^{\infty} \gamma^k P(s_k = s \mid s_o, \pi)$ is used. The infinite-horizon average return can be seen as the infinite-horizon discounted return model where the discount factor approaches 1 (Tsitsiklis and Roy, 1999).

A measure for the quality of the current state or current state-action pair under policy π can be given by a *state value function* $V^\pi(s) : \mathcal{S} \to \mathbb{R}$ or a *state-action value function* $Q^\pi(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. These are also called *V-functions* and *Q-functions*. For the infinite-horizon discounted return these are:

$$V^\pi(s) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} \;\middle|\; s_0 = s \right\}, \tag{2-9}$$

$$Q^\pi(s,a) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} \;\middle|\; s_0 = s, a_0 = a \right\}. \tag{2-10}$$

For a single start state s_0 the value of the begin state is equal to the return $V^\pi(s_0) = \rho(\pi)$. The value functions satisfy the Bellman Equations (Sutton and Barto, 1998):

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \left(\mathcal{R}(s,a,s') + \gamma V^\pi(s') \right), \tag{2-11}$$

$$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \left(\mathcal{R}(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(s',a') Q^\pi(s',a') \right). \tag{2-12}$$

For the average return setting the value functions under policy π are:

$$V^\pi(s) = E\left\{ \sum_{k=0}^{\infty} (r_{k+1} - \rho(\pi)) \;\middle|\; s_0 = s \right\}, \tag{2-13}$$

$$Q^\pi(s,a) = E\left\{ \sum_{k=0}^{\infty} (r_{k+1} - \rho(\pi)) \;\middle|\; s_0 = s, a_0 = a \right\}. \tag{2-14}$$

The corresponding Bellman equations are given by:

$$V^\pi(s) + \rho(\pi) = E\left\{ \mathcal{R}(s,a,s') + V^\pi(s') \right\}, \tag{2-15}$$

$$Q^\pi(s,a) + \rho(\pi) = E\left\{ \mathcal{R}(s,a,s') + Q(s',a') \right\}, \tag{2-16}$$

where the actions a and a' are generated by π. When in each state the optimal action or action distribution, that is following the optimal policy π^*, is taken, the value functions are maximized. The optimal state value function $V^*(s)$ and action-value function $Q^*(s,a)$ are defined as:

$$V^*(s) = \max_\pi V^\pi(s), \tag{2-17}$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a). \tag{2-18}$$

$$\tag{2-19}$$

A policy that always takes the action with the highest expected return is called a *greedy policy*. During learning, an agent cannot follow a completely greedy policy, because the agent needs to explore unseen parts of the state-action space. This can be ensured by sometimes taking exploratory actions instead of greedy actions. A simple exploration method is *ϵ-greedy action selection*: with probability $1-\epsilon$ (where $\epsilon \in (0,1)$) the greedy action is taken and with probability ϵ a random action is chosen from $\mathcal{A}$ (Sutton and Barto, 1998). Other exploration methods are *Boltzmann exploration* and a *Gaussian policy*. A problem that arises is the balance between exploration and obtaining the highest possible return; exploitation. When insufficiently many states-action pairs have been visited the greedy action can be suboptimal, while taking an exploration step when the optimal policy has been learned is also suboptimal.

2-2 Solving the Reinforcement Learning Problem

Solution methods for the RL problem can be roughly divided into three groups (Grondman et al., 2011a; Marbach and Tsitsiklis, 1998) based on the action selection and the policy update. *Critic-only* methods select actions based on the value function. *Actor-only* methods search for the optimal policy directly in the policy space. *Actor-critic methods* combine the advantages of both methods. This section first introduces critic-only algorithms. Thereafter, actor-only and actor-critic methods are treated.

2-2-1 Critic-Only Methods

Critic-only methods learn an estimate of the value function by solving the Bellman equation (2-11) or (2-12) recursively. First, model based critic-only methods are treated. Then model-free techniques are introduced, followed by methods to increase the learning speed.

Dynamic Programming

Dynamic programming methods solve MDP's by finding the value function recursively using a model. The Bellman equation is used as an update rule. In *value iteration* the value of a state action pair is updated according to:

$$Q_{k+1}(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma \max_{a'} Q_k(s',a')]. \tag{2-20}$$

Policy iteration evaluates the value function of a policy π using (2-12). After the policy evaluation the policy improvement step creates a new policy that is greedy in the learned value function. This process is repeated until the optimal policy has been found. In generalized policy iteration (GPI) the switch between policy evaluation and policy improvement and vice versa is made before the current step has converged.

Temporal Difference Methods

One of the most significant ideas in RL is temporal difference (TD) learning (Sutton, 1988). The TD error δ is the error between the expected and the received return and is defined for

the infinite-horizon discounted and average return setting as respectively:

$$\delta_{k+1} = r_{k+1} + \gamma V(s_{k+1}) - V(s_k), \tag{2-21}$$
$$\delta_{k+1} = r_{k+1} - \hat{\rho} + V(s_{k+1}) - V(s_k). \tag{2-22}$$

The value function for a policy can be computed incrementally with Sutton's TD algorithm, which incorporates policy iteration (Sutton, 1988):

$$V_{k+1}(s_k) = V_k(s_k) + \alpha_k \delta_{k+1}. \tag{2-23}$$

The parameter $\alpha_k \in (0, 1]$ is the *learning rate* at time instant k. It is a weighting between the new and the old estimate of the value function. The update process (2-23) converges when the learning rate schedule satisfies the *Robbins-Monro* conditions (Robbins and Monro, 1951; Szepesvári, 2010),

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \alpha_k^2 < +\infty, \qquad \alpha_k \geq 0 \quad \forall k,$$

and all state-action pairs are visited infinitely many times. The most well-known TD-algorithms are *Sarsa* and *Q-learning*. Sarsa learns the Q-function of the policy followed; it is an *on-policy* method. It updates according to:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k[r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1}) - Q_k(s_k, a_k)]. \tag{2-24}$$

Off-policy methods learn a value function for a policy that is different from the policy used to generate the actions; the target policy is different from the behavior policy. Q-learning is an off-policy method that learns the Q-function of the greedy policy,

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k[r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k)]. \tag{2-25}$$

For a more thorough explanation of critic-only algorithms, the interested reader is referred to (Buşoniu et al., 2010; Sutton and Barto, 1998; Szepesvári, 2010).

Using Samples More Efficiently The TD-methods described above use each sample only once to update the Q-function. Learning can be accelerated by using the samples more efficiently. The *eligibility trace* $e : \mathcal{S} \times \mathcal{A} \to [0, \infty)$ allows the current reward r_{k+1} to update the Q-function of recently visited states. The update rule of a TD-algorithm with eligibility traces is denoted by:

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha_k \delta_k e_{k+1}(s, a) \ , \qquad \forall s, a, \tag{2-26}$$

with $e_0 = 0$. The most often used method to determine the eligibility trace is by *replacing traces* (Babuška, 2010):

$$e_{k+1}(s, a) = \begin{cases} 1 & \text{if } (s, a) = (s_k, a_k) \\ \lambda\gamma e_k(s, a) & \text{if } (s, a) \neq (s_k, a_k) \end{cases} \qquad \forall (s, a), \tag{2-27}$$

where $\lambda \in [0, 1]$ is the *trace-decay parameter*. Another method in which samples can be used more efficiently is by recording transitions $(s_k, a_k, r_{k+1}, s_{k+1}, a_{k+1})$ and replaying them in the update equation. This *experience replay* has shown to increase learning speed tremendously (Adam et al., 2012; Wawrzynski, 2009).

Generalization

Sarsa and Q-learning use a tabular representation of the value function. A table for a Q-function has size $|\mathcal{S}| \times |\mathcal{A}|$ and Q-values become accurate if all state-action pairs are visited a large number of times. For high-dimensional state and action spaces this becomes impractical. Continuous state and action spaces cannot be implemented while many real world applications demand continuous control (Gaskett et al., 1999). Furthermore, knowledge about similar states is not used. These problems can be overcome by *generalization* methods. Generalization can be introduced by *function approximation*. The value function approximated by a parametric function approximator is denoted by:

$$V_\theta(s) = f(s,\theta) \qquad \text{or} \qquad Q_\theta(s,a) = f(s,a,\theta), \tag{2-28}$$

where θ is the parameter to be learned. This can be done by using a gradient descent method (Sutton and Barto, 1998):

$$\theta_{k+1} = \theta_k + \alpha_k \delta_{k+1} \frac{\partial V_\theta(s_k)}{\partial \theta}, \tag{2-29}$$

where α_k has to satisfy the Robbins Monro conditions.

Linearly Parameterized Approximators An approximation of a value function that is linear in the parameters (LIP) is denoted by:

$$V_\theta(s) = \theta^T \phi(s) \qquad \text{or} \qquad Q_\theta(s,a) = \theta^T \phi(s,a), \tag{2-30}$$

where $\theta \in \mathbb{R}^p$ is parameter vector and $\phi(s) : \mathcal{S} \to \mathbb{R}^p$ and $\phi(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^p$ are features. A single feature i is denoted by ϕ_i. Features are also called basis functions (BFs) (Buşoniu et al., 2010). They are a useful tool to incorporate a priori knowledge of the designer about the problem (Bertsekas and Tsitsiklis, 1995). Typical BFs are binary features constructed with coarse or tile coding, triangular, polynomial and radial basis functions (RBFs). In this thesis Gaussian RBFs are used, because they can represent a smooth value function and have shown to work better than triangular and polynomial BF's. Normalized Gaussian RBFs are denoted by:

$$\phi_i(s) = \frac{\bar{\phi}_i(s)}{\sum_{i'=1}^{n} \bar{\phi}_{i'}(s)}, \qquad \bar{\phi}_i(s) = \exp\left(-\frac{1}{2}(s-c_i)^T B_i^{-1}(s-c_i)\right), \tag{2-31}$$

where $\bar{\phi}_i(s)$ is the non-normalized feature. The center of the RBF is given by the vector $c_i = [c_{i,1}, c_{i,2}, \ldots, c_{i,n}]^T \in \mathbb{R}^n$ where n is the dimension of the state. The width is determined by the symmetric positive definite matrix $B \in \mathbb{R}^{n \times n}$. More information about the construction and choice of BF's can be found in Buşoniu et al. (2010); Sutton and Barto (1998).
A LIP value function approximation is often used because the theoretical properties are easy to analyze (Buşoniu et al., 2010). For approximators that are LIP. convergence is guaranteed for on-policy methods (Tsitsiklis and Roy, 1997). Moreover, θ can be found using least-squares by Least-Squares Temporal Difference (LSTD). The LSTD algorithm with eligibility traces, referred to as LSTD(λ), is denoted by (Boyan, 2002):

$$e_{k+1} = \lambda e_k + \phi(s_k), \tag{2-32}$$

$$A_{k+1} = A_k + e_k(\phi(s_k) - \gamma\phi(s_{k+1}))^T, \tag{2-33}$$

$$b_{k+1} = b_k + e_k r_{k+1}, \tag{2-34}$$

$$\theta_{k+1} = A_{k+1}^{-1} b_{k+1}, \tag{2-35}$$

where $e_0 = \phi(s_0)$, $b_0 = 0$ and A_0 is chosen as a small invertible matrix.

Local Linear Regression is a non-parametric method that approximates a nonlinear function $y = f(x)$, with input vector $x \in \mathbb{R}^n$ and output vector $y \in \mathbb{R}^m$ by fitting a locally affine model to stored input-output data. The output for a given input x is then approximated by:

$$\hat{y} = \beta \begin{bmatrix} x \\ 1 \end{bmatrix}, \tag{2-36}$$

where β is the model parameter matrix. During learning, observations of input-output data, called samples, are collected in the memory. The samples are column vectors $\varsigma_i = [x_i^T \mid y_i^T]^T$, with $i = 1 \dots N$. The samples are stored in the columns of the memory matrix M, of size $(m+n) \times N$. Outdated samples have to be replaced with new samples.
To obtain the process model β first the K nearest samples in the memory are selected, according to a (weighted) distance metric (Grondman et al., 2012b). The input and output data of the nearest neighbors are collected in the matrices:

$$Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_K \end{bmatrix}, \qquad X = \begin{bmatrix} x_1 & x_2 & \cdots & x_K \\ 1 & 1 & \cdots & 1 \end{bmatrix}. \tag{2-37}$$

The model parameter β is found by minimizing the squared error $\varepsilon = |Y - \beta X|^2$. For example, this can be done with the right pseudo-inverse:

$$\beta = YX^T(XX^T)^{-1}. \tag{2-38}$$

2-2-2 Actor-Only Methods

Instead of using a value function to determine the action, actor-only methods use a parameterized policy to select the actions. The parameterized policy is denoted by π_ϑ, with the policy parameter vector $\vartheta \in \mathbb{R}^q$. Many actor-only methods try to estimate the gradient $\frac{\partial \rho}{\partial \vartheta}$, in order to optimize the policy vector by the following gradient ascent update rule:

$$\vartheta_{k+1} = \vartheta_k + \alpha \frac{\partial \rho}{\partial \vartheta}. \tag{2-39}$$

The gradient can be found in multiple ways. Policy Gradients with Parameter-Based Exploration (PGPE) (Sehnke et al., 2010) estimates this gradient by parameter perturbation. If the policy π_ϑ is differentiable in ϑ, the performance gradient is

$$\frac{\partial \rho}{\partial \vartheta} = \frac{\partial \rho}{\partial \pi} \frac{\partial \pi}{\partial \vartheta}. \tag{2-40}$$

This gradient can be estimated from a single simulation of the system using the *likelihood ratio* method. REINFORCE (Williams, 1992) and Gradients of Partial Observable Markov Decision Processes (GPOMDP) (Baxter and Bartlett, 2001) are actor-only methods that use (2-40) to update their policy. Actor-only methods tend to suffer from high variance in the policy gradient estimate. This leads to slow learning.

2-2-3 Actor-Critic Methods

This section first introduces the concept of the actor-critic framework. Thereafter two different policy gradient methods are discussed.

Actor-critic methods combine actor- and critic-only methods by having an element to represent the value function (the critic) and an element representing the policy (the actor). The general overview of the actor-critic method is given in Figure 2-2. Because both the value

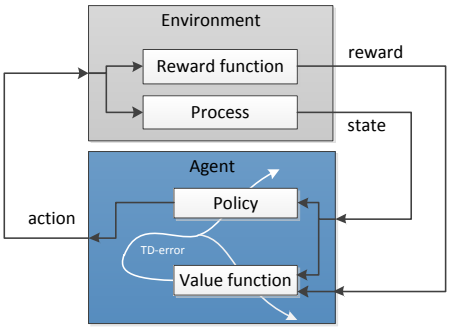


Figure 2-2: The architecture of the actor-critic framework

function V and the policy π can be parameterized functions (V_θ and π_ϑ), actor-critic algorithms can cope with continuous state and action spaces. The critic uses the state and the reward from the environment to learn the value function of the followed policy. With this value function it evaluates the actions taken by the actor. This evaluation is used for the actor update. The critic can be used to reduce the variance in the actor update. In this thesis both the actor and the critic function approximator are LIP. The policy of choice is a Gaussian policy:

$$a \sim \underbrace{\vartheta^T \phi_{\mathrm{a}}(s)}_{\mu(s)} + \underbrace{\mathcal{N}(0, \sigma^2)}_{\varepsilon_k}. \tag{2-41}$$

The action a is the sum of the average action $\mu(s)$ and the exploration term ε_k drawn from the normal distribution with standard deviation σ. The subscript a denotes that it is the actor BF.
The most basic actor-critic algorithm used in this thesis is Standard Actor-Critic (SAC) (Grondman et al., 2011b). The critic learns the state value function $V_\theta(s)$ using a TD-update. The value function is updated according to:

$$\theta_{k+1} = \theta_k + \alpha \delta_{k+1} \frac{\partial \theta V_{\theta_k}(s_k)}{\partial \theta}. \tag{2-42}$$

The actor is updated with:

$$\vartheta_{k+1} = \vartheta_k + \alpha_a \delta_{k+1} \varepsilon_k \frac{\partial \pi(s,a)}{\partial \vartheta}. \tag{2-43}$$

If an exploratory move is successful (unsuccessful) the TD-error is positive (negative) and the actor is steered towards (away from) taking this action. In this way an estimate of the policy gradient (2-39) is made.

Policy Gradient Theorem

Another method to estimate the policy gradient is given by the policy gradient theorem (Konda and Tsitsiklis, 2003; Sutton et al., 1999). The policy gradient for both discounted and the average return setting is

$$\frac{\partial \rho}{\partial \vartheta} = \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} Q^{\pi}(s,a) \, \mathrm{d}a \, \mathrm{d}s, \tag{2-44}$$

with the steady state distribution $d^{\pi}(s)$ defined as in Section 2-1-2. The main advantage of (2-44) is that there is no term $\frac{\partial d^{\pi}(s)}{\partial \vartheta}$ present. This implies that given a data set obtained by following π, $\int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} Q^{\pi}(s,a) \, \mathrm{d}a$ is an unbiased estimate of $\frac{\partial \rho}{\partial \vartheta}$. It furthermore shows the connection between the policy gradient and the critic function.
Another advantage of the Policy Gradient Theorem is that it also applies to parameterized critic functions. Let $f_w(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be the approximation of the state-action value function $Q^{\pi}(s,a)$, parameterized by w. Most function approximators try to minimize the mean squared error (MSE), where w is updated according to $\Delta w_k \propto \frac{\partial}{\partial w}[Q^{\pi}(s,a) - f_w(s,a)] \propto [Q^{\pi}(s,a) - f_w(s,a)] \frac{\partial f_w(s,a)}{\partial w}$. This will converge to a (local) optimum satisfying:

$$\int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \pi_{\vartheta}(s,a) \left[Q^{\pi}(s,a) - f_w(s,a) \right] \frac{\partial f_w(s,a)}{\partial w} \, \mathrm{d}a \, \mathrm{d}s = 0. \tag{2-45}$$

When $f_w(s,a)$ satisfies (2-45) and the compatibility condition

$$\frac{\partial f_w(s,a)}{\partial w} = \frac{\partial \ln \pi_{\vartheta}(s,a)}{\partial \vartheta}, \tag{2-46}$$

substitution of the right hand side of (2-46) in (2-45) gives:

$$\int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} \left[Q^{\pi}(s,a) - f_w(s,a) \right] \, \mathrm{d}a \, \mathrm{d}s = 0. \tag{2-47}$$

The approximation error is orthogonal to the gradient of the policy parametrization. Subtracting (2-47) from (2-44) gives

$$\frac{\partial \rho}{\partial \vartheta} = \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} f_w(s,a) \, \mathrm{d}a \, \mathrm{d}s. \tag{2-48}$$

When the value function approximation satisfies (2-45) and (2-46), an unbiased estimate of the policy gradient is obtained by (2-48). It is noteworthy that f_w has zero mean with respect to the action distribution, that is

$$\int_{\mathcal{A}} \pi(s,a) f_w(s,a) \, \mathrm{d}a = 0 \qquad \forall s \in \mathcal{S}. \tag{2-49}$$

Thus it is better to think of f_w as an *advantage function*: $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$ (Peters and Schaal, 2008a; Sutton et al., 1999). Since $\int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} \mathrm{d}a = 0 \quad \forall s \in \mathcal{S}$, an arbitrary function $b : \mathcal{S} \to \mathbb{R}$ can be added without affecting the results of (2-48). The function $b(s)$, in (Peters and Schaal, 2008a) referred to as a baseline, can have a big influence on the variance of the gradient estimators (Sutton et al., 1999). When the baseline is a parameterized function b_v with parameter vector v, the value function Q_{θ} can be chosen as:

$$Q_{\theta} = f_w(s,a) + b_v(s), \tag{2-50}$$

with $\theta = [w^T \; v^T]^T$. An important property of the policy gradient methods is that convergence to a policy is guaranteed for approximate value functions satisfying the compatibility condition (Sutton et al., 1999).

Natural Gradient

The previously described policy gradient methods use a regular gradient. The use of the regular gradient has a drawback that it is different for every parameterization of the policy π_ϑ. Often the cost functions that have to be optimized are not Euclidean (Amari and Douglas); for instance a robot arm where the position of a gripper and the reward are determined by the angle of its joints. The state space is then not Euclidean, but curved, or *Riemannian* in many cases (Amari, 1998). Therefore the ordinary, or *vanilla gradient*, does not give the steepest direction of the cost function, but the *natural gradient* does. It is a covariant gradient (Kakade, 2001a): it is independent of the parametrization of the space. The natural gradient of cost function $\rho(\vartheta)$ is a linear transformation of the vanilla gradient given by:

$$\widetilde{\frac{\partial \rho(\vartheta)}{\partial \vartheta}} = G^{-1}(\vartheta)\frac{\partial \rho(\vartheta)}{\partial \vartheta}, \tag{2-51}$$

where $G(\vartheta)$ is the *Riemannian metric tensor*, a positive-definite matrix of size $n \times n$ that characterizes the intrinsic curvature of the n-dimensional space (Amari and Douglas). The Riemannian metric tensor in the parameter space of a statistical model is the Fisher Information Matrix (FIM).

A Natural Actor-Critic Algorithm

The Natural Actor-Critic (NAC) with LSTD-Q(λ) algorithm incorporates the natural gradient and the policy gradient theorem (Peters and Schaal, 2008a). The compatible approximation of the Q-function is :

$$f_w^\pi(s,a) = \frac{\partial \ln \pi(s,a)}{\partial \vartheta}^T w. \tag{2-52}$$

The gradient of the expected return function can be found by plugging (2-52) into (2-48):

$$\frac{\partial \rho}{\partial \vartheta} = \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \pi_\vartheta(s,a) \frac{\partial \ln \pi(s,a)}{\partial \vartheta} \frac{\partial \ln \pi(s,a)}{\partial \vartheta}^T w \, \mathrm{d}a \, \mathrm{d}s \tag{2-53}$$

$$= F_\vartheta w. \tag{2-54}$$

It is proven by Peters and Schaal (2008a) that the integral represented by F_ϑ is the FIM. Therefore the natural gradient is given by:

$$\widetilde{\frac{\partial \rho(\vartheta)}{\partial \vartheta}} = G^{-1}(\vartheta) F_\vartheta w = w. \tag{2-55}$$

The policy can thus be improved by $\vartheta_{k+1} = \vartheta_k + \alpha_a w$. To solve this equation the advantage function f_w has to be learned. The function cannot be learned by TD bootstrapping methods without knowledge of the value function (Peters and Schaal, 2008a). It can be derived by using the Bellman equation (2-12):

$$Q^\pi(s,a) = A^\pi(s,a) + V^\pi(s). \tag{2-56}$$

Peters and Schaal (2008a) use a value function approximation that is LIP:

$$Q^\pi(s,a) = \frac{\partial \ln \pi(s,a)}{\partial \vartheta}^T w + \theta^T \phi_{\mathrm{c}}(s_k). \tag{2-57}$$

The Q-function is learned using an adapted verion of LSTD(λ), hence called LSTD-Q(λ). It has the updates:

$$\hat{\phi}_k = [\phi_c(s_k)^T \quad \frac{\partial \ln \pi(s_k, a_k)}{\partial \vartheta}^T]^T, \tag{2-58}$$

$$\tilde{\phi}_k = [\phi_c(s_{k+1})^T \quad 0^T]^T, \tag{2-59}$$

$$e_{k+1} = \lambda e_k + \hat{\phi}_k, \tag{2-60}$$

$$A_{k+1} = A_k + e_{k+1}(\hat{\phi}_k - \gamma\tilde{\phi}_k)^T, \tag{2-61}$$

$$b_{k+1} = e_{k+1} r_{k+1}, \tag{2-62}$$

$$\begin{bmatrix}\theta^T & w^T\end{bmatrix}^T = A_{k+1}^{-1} b_{k+1}, \tag{2-63}$$

in which 0 is the zero vector. When the value function has converged, the actor has to be updated. The value function is considered to be converged when the angle between consecutive gradients is smaller than a threshold value ϵ. Because NAC is an on-policy method, information in the critic gets outdated after the actor update. Therefore a forgetting factor $\eta_c \in [0, 1]$ is introduced. After each actor update, the critic discounts old information according to:

$$A_{k+1} \leftarrow \eta_c A_{k+1} \ , \qquad b_{k+1} \leftarrow \eta_c b_{k+1} \ . \tag{2-64}$$

The complete algorithm is given in Algorithm 2.1.

Algorithm 2.1 Natural actor critic with LSTD-Q(λ)

Input: : α_a, η_c, ϵ, λ, ϑ_0, θ_0, features ϕ, parametrized policy π_ϑ and the derivative $\frac{\partial \ln \pi(s,a)}{\partial \vartheta}$

initialize state s_0, A_0, $b_{k+1} = 0$, $e_{k+1} = 0$
for every step $k = 1, 2, \ldots$ **do**
  select and implement action using π_{ϑ_k}
  observe next state s_{k+1} and reward r_{k+1}
  LSTD-Q
    $\hat{\phi}_k = [\phi_c(s_k)^T \ \frac{\partial \ln \pi(s_k,a_k)}{\partial \vartheta}^T]^T$
    $\tilde{\phi}_k = [\phi_c(s_{k+1})^T \ 0^T]^T$
  $e_{k+1} = \lambda e_k + \hat{\phi}_k$, $A_{k+1} = A_k + e_{k+1}(\hat{\phi}_k - \gamma\tilde{\phi}_k)^T$, $b_{k+1} = b_k + e_{k+1} r_k$
  $[\theta_{k+1}^T \ w_{k+1}^T]^T = A_{k+1}^{-1} b_{k+1}$
  Actor
  if gradient estimate is accurate $\measuredangle(w_k, w_{k-1}) \leq \epsilon$ **then**
    $\vartheta_{k+1} = \vartheta_k + \alpha_a w_{k+1}$
    $e_{k+1} \leftarrow \eta_c e_{k+1}$, $A_{k+1} \leftarrow \eta_c A_{k+1}$, $b_{k+1} \leftarrow \eta_c b_{k+1}$
  end if
end for

2-3 Conclusion

The reinforcement learning framework allows an agent to become proficient in a task in an unknown environment, using the observed states, actions and rewards as information. The adopted formal model is the Markov decision process. Three classes of solution methods have been treated:

- **Critic-only** methods use a value function for action selection.
- **Actor-only** algorithms search directly in the policy space for the optimal policy.
- **Actor-critic** methods combine the actor- and critic-only methods by having an element that represents the controller (the actor) and an element that learns a value function (the critic).

Of the above methods, actor-critic (AC) is investigated first, because it has promising properties for robotic applications: it can be easily applied to continuous state and action spaces, and has been proven to be able to learn complex tasks more quickly than actor-only methods (Peters and Schaal, 2008b). In this thesis approximate policy gradient methods are used because they have been successfully applied to difficult learning problems (Kohl and Stone, 2004; Peters and Schaal, 2008b) and convergence for these methods is guaranteed.

Chapter 3

Levenberg-Marquardt

An unresolved issue for policy gradient methods is the choice of the step size in the actor update. Descent algorithms in classical optimization face the same problem. The step size is a trade-off between a slow accurate estimate of the optimum and a quick crude estimation. A very powerful method to determine the step size is the Levenberg Marquardt (LM) algorithm (Marquardt, 1963). The algorithm has a proven track record in both classical optimization (Verhaegen and Verdult, 2007) and learning (Ni et al., 2011). The gradient and the Hessian are used to determine the descent step. Advantages of the LM algorithm are:

- It has good convergence properties (Hagan and Menhaj, 1994)
- The algorithm can cope with stochasticity as faced in learning (Dias et al., 2006; Ni et al., 2011).

Because of these good properties, the LM algorithm is considered to be the most promising method to determine the actor step size. Three chapters are devoted to devising an LM actor step size method. In this chapter an introduction to the LM algorithm is given. Methods to obtain the gradient and the Hessian are discussed in Chapter 4 and 5, respectively.

The Algorithm

The objective of the LM method is to optimize the unconstrained optimization problem:

$$\min_x f(x), \tag{3-1}$$

with $f : x \to \mathbb{R}$ and $x \in \mathbb{R}^n$. The LM method is developed to be an optimum interpolation between the gradient descent method and the Newton method (Marquardt, 1963). The Newton method assumes that the objective function $f(x)$ can be approximated by a second order Taylor series expansion around an evaluated point x (Luenberger and Ye, 2008):

$$f(x + \Delta x) \approx \frac{1}{2}\Delta x^T H(x)\Delta x + J(x)\Delta x + f(x), \tag{3-2}$$

where$H(x) \in \mathbb{R}^{n\times n}$ is the Hessian and $J(x) \in \mathbb{R}^n$ the Jacobian at x. If the Hessian is positive definite and the Taylor series expansion is accurate, it is reasonable to assume that the minimum of the approximation is close to the true minimum. The minimum of the approximation is found by setting the derivative of Equation (3-2) to zero. The *Newton step* is the step towards this minimum

$$x_{k+1} = x_k - H(x_k)^{-1}J(x). \tag{3-3}$$

The Newton algorithm has the following advantages (Boyd and Vandenberghe, 2004):

- It is invariant to linear and affine changes of the objective function.
- It has good convergence properties. For convex functions it converges quadratically near the optimum x^*.
- It is a parameter-free algorithm. No scaling parameters, learning rates, or meta-parameters have to be set.
- The Newton's algorithm scales well with the dimension of x.

For general non-convex problems the Newton method has two main disadvantages:

- A nearly singular Hessian results in unreasonably large steps.
- When the Hessian is negative definite, the calculated update step is in an ascending direction and is therefore of no use.

LM cures these problems by using a modified positive definite Hessian. The LM step is given by

$$x_{k+1} = x_k - \hat{H}(x)^{-1}J(x_k), \tag{3-4}$$

$$\hat{H}(x_k) = H(x_k) + \kappa_k DD^T, \tag{3-5}$$

in which $\kappa_k \geq 0$ is a damping parameter and D a nonsingular scaling matrix. LM is an interpolation between gradient descent methods and the Newton method; for small values of κ the solution is close to the Newton method, while for large values and $D = I$ the solution approximates the gradient descent method with step size $1/\kappa$. The size and the direction of the step are altered by the choice of κ. There are different methods to choose the scaling factor κ and the scaling matrix D. These methods will be discussed hereafter.

Scaling Matrix

In the literature different compositions of the scaling matrix $D \in \mathbb{R}n \times n$ are given. In van den Boom and de Schutter (2009) and Dias et al. (2006) the identity matrix is used. In Moré (1977) a diagonal matrix is chosen where the diagonal components $[d_{1,k}, \ldots, d_{n,k}]$ are given by:

$$d_{i,k} = \max(d_{i,k-1}, \|\partial_i f(x_k)\|), \tag{3-6}$$

with $d_{i,0} = \|\partial_i f(x_o)\|$.

Damping Factor

There are two different approaches to choose the damping factor κ: line search and trust region (Ranganathan, 2004).

- *Line search.* This approach views LM as a blend of the Newton and the gradient descent method. When an update step is descending (successful) it is assumed that the Hessian estimate is accurate enough and κ can be decreased. An unsuccessful step is retracted and κ is increased to decrease the step size.
- *Trust region.* The modified Hessian approximation(3-5) will only be valid in the neighborhood of the current state x_k. This imposes the constraint on the step Δx:

$$\|D\Delta x\| \leq Z, \tag{3-7}$$

where Z is a step bound that depends on the ratio between the actual reduction of the objective function and the predicted reduction. This ratio is denoted by the variable ϱ. In (Moré, 1977) ϱ is calculated as follows:

$$\varrho(\Delta x) = \frac{\|f(x)\|^2 - \|f(x+\Delta x)\|^2}{\|f(x)\|^2 - \|f(x) + J\Delta x\|^2}. \tag{3-8}$$

The step bound is adjusted according to:

$$Z_{k+1} = \begin{cases} 2\|D_k \Delta x_k\| & \text{if } \varrho_k > 0.75 \\ \left[\frac{1}{10}Z_k \ , \ \frac{1}{2}Z_k\right] & \text{if } \varrho_k < 0.25 \\ Z_k & \text{otherwise,} \end{cases} \tag{3-9}$$

where the initial step bound Z_0 is set to a bound in which the Taylor series expansion is presumed to be valid. The current method to determine ϱ and κ is (Dias et al., 2006):

$$\varrho(\Delta x) = \frac{f(x) - f(x+\Delta x)}{f(x) - L(\Delta x)} \tag{3-10}$$

$$\kappa_{k+1} = \begin{cases} \kappa_k/2 & \text{if } \varrho_k \geq 0.75 \\ 2\kappa_k & \text{if } \varrho_k \leq 0.25 \\ \kappa_k & \text{otherwise} \end{cases} \tag{3-11}$$

in which $L(\Delta x)$ is the second order Taylor series expansion (3-2) of f around x.

For both methods the literature does not provide an initial scaling factor κ. It should be chosen at least large enough to render $\hat{H}_0$ positive definite.

Remarks

In this chapter the LM algorithm has been briefly introduced as a descent method, because this form is most often adopted in optimization literature. Because RL algorithms try to maximize the return, the algorithms in the rest of this thesis are treated as ascent algorithms. For the LM method this implies that the Hessian has to be negative definite instead of positive definite.

Chapter 4

Vanilla Actor-Critic

The Levenberg Marquardt (LM) algorithm uses the vanilla gradient in its update. In this chapter the Policy Gradient Theorem is used to find the vanilla gradient for a Gaussian policy. This vanilla policy gradient is used in a novel vanilla actor-critic method that is based on Natural Actor-Critic (Peters and Schaal, 2008a). The learning behavior of this algorithm is tested on the inverted pendulum swing-up task. Furthermore, the algorithm is compared with natural gradient methods to provide more insight in the differences between these two gradient ascent methods.

4-1 Vanilla Policy Gradient

The vanilla gradient is obtained with the Policy Gradient Theorem (Section 2-2-3). The policy gradient for a parameterized value function is given by:

$$\frac{\partial \rho}{\partial \vartheta} = \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} f_w(s,a) \, \mathrm{d}a \, ds, \tag{4-1}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $d^{\pi}(s)$ the steady state distribution and $f_w(s,a)$ the compatible advantage function approximation. For a Gaussian policy, that select actions according to $a \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu = \vartheta^T \phi_a(s)$, the compatible advantage function approximation is:

$$f_w(s,a) = \left(\frac{a-\mu}{\sigma^2} \phi_a(s) \right)^T w, \tag{4-2}$$

where the advantage function parameter w is the natural gradient (Peters and Schaal, 2008a). The derivation is given in Appendix A-1. It is proven in Appendix A-1 that with the compatible value function approximation the vanilla gradient is:

$$\frac{\partial \rho}{\partial \vartheta} = \int_{\mathcal{S}} d^{\pi}(s) \frac{1}{\sigma^2} \phi(s) \phi(s)^T \mathrm{d}s \cdot w \tag{4-3}$$

$$= F_{\vartheta} w. \tag{4-4}$$

The integral $\int_{\mathcal{S}} d^{\pi}(s)\frac{1}{\sigma^2}\phi(s)\phi(s)^T \mathrm{d}s$ is the Fisher Information Matrix (FIM) F_{ϑ} (Peters and Schaal, 2008a). The vanilla gradient is a linear transformation of the natural gradient. The FIM is policy dependent through the steady state distribution $d^{\pi}(s)$. Hence the FIM needs to be relearned when the policy is updated. An unbiased estimate of the FIM can be acquired by :

$$F_{\vartheta} \approx \frac{1}{N\sigma^2}\sum_{i=1}^{N}\phi_a(s_i)\phi_a(s_i)^T = \frac{1}{N}\Phi, \tag{4-5}$$

where $s_1 \dots s_N$ is a trajectory generated under π and $\Phi = \frac{1}{\sigma^2}\sum_{i=1}^{N}\phi_a(s_i)\phi_a(s_i)^T$. Although the estimator is unbiased, it takes many samples to acquire a good estimate, because the steady state distribution needs to be properly captured. If the steady state distribution $d^{\pi}(s)$ changes slowly for small policy updates it seems wasteful to discard the FIM after each actor update. Past estimates of the FIM can be used in a discounted way to form the current estimate of the FIM. The used update scheme for the FIM is:

1. Initialize $\Phi = 0$ and $N = 0$.
2. After selecting an action, the matrix summation Φ and the counter N are updated.

$$\Phi \leftarrow \Phi + \frac{1}{\sigma^2}\phi_a(s_k)\phi_a(s_k)^T \qquad N \leftarrow N + 1. \tag{4-6}$$

3. If the FIM is requested, it is calculated with:

$$F \leftarrow \frac{1}{N}\Phi. \tag{4-7}$$

4. After each actor update Φ and N need to be discounted:

$$\Phi \leftarrow \eta_F \Phi \ , \qquad N \leftarrow \eta_F N \ , \tag{4-8}$$

   where $\eta_F \in [0, 1]$ is the discount factor of the FIM.

By plugging the learned FIM in Equation (4-4) the vanilla policy gradient of a Gaussian policy is found. To make a vanilla actor-critic algorithm the natural gradient w needs to be learned. Because the LSTD critic in NAC (Peters and Schaal, 2008a) provides a very easy and fast method to estimate the natural gradient, the same critic is used in Vanilla Actor-Critic (VAC).
A number of modifications have to be made to create VAC from NAC. First, the four steps of the FIM update need to be incorporated in the algorithm. Second, the VAC actor update is:

$$\vartheta_{k+1} = \vartheta_k + \alpha_{a,k} F w_{k+1}. \tag{4-9}$$

A fifth modification has to be made to maintain the numerically invertibility of the A-matrix of LSTD-Q. NAC uses a forgetting factor $\eta_c \in [0, 1]$ to discount old information of the critic in the A-matrix and b-vector after each actor update (see Section 2-2-3). The A-matrix is initialized with a small invertible matrix A_0. At each actor update a fraction of the initial matrix A_0 is lost by the discounting. To keep A numerically invertible this fraction has to be added:

$$A_{k+1} \leftarrow \eta_c A_{k+1} + (1 - \eta_c)A_0. \tag{4-10}$$

The complete version of the VAC algorithm is given in Algorithm 4.1.

Algorithm 4.1 Vanilla Actor Critic with a Gaussian policy

Input: : α_a, η_c,η_F, ϵ, λ, γ, ϑ_0, θ_0, features ϕ, Gaussian policy π_ϑ

initialize state s_0, A_0, b_0, $e_{k+1} = 0$, $\Phi = 0$, $N = 0$
for every step $k = 1, 2, \ldots$ **do**
  draw action $a_k \sim \mathcal{N}(\vartheta^T \phi_a(s_k), \sigma^2)$
  $\Phi \leftarrow \Phi + \frac{1}{\sigma^2}\phi_a(s_k)\phi_a(s_k)^T$, $N \leftarrow N + 1$
  observe next state s_{k+1} and reward r_{k+1}
  LSTD-Q
    $\hat{\phi}_k = \begin{bmatrix}\phi_c(s_k)^T & \frac{a-\mu}{\sigma^2}\phi_a(s)^T\end{bmatrix}^T$
    $\tilde{\phi}_k = \begin{bmatrix}\phi_c(s_{k+1})^T & 0^T\end{bmatrix}^T$
  $e_{k+1} = \lambda e_k + \hat{\phi}_k$, $A_{k+1} = A_k + e_{k+1}(\hat{\phi}_k - \gamma\tilde{\phi}_k)^T$, $b_{k+1} = b_k + e_{k+1}r_{k+1}$
  $\begin{bmatrix}\theta_{k+1}^T & w_{k+1}^T\end{bmatrix}^T = A_{k+1}^{-1} b_{k+1}$
  Actor
  if gradient estimate is accurate $\measuredangle(w_k, w_{k-1}) \leq \epsilon$ **then**
    $\mathrm{F} = \frac{1}{N}\Phi$
    $\vartheta_{k+1} = \vartheta_k + \alpha_a F w_{k+1}$
    $e_{k+1} \leftarrow \eta_c e_{k+1}$, $A_{k+1} \leftarrow \eta_c A_{k+1}$, $b_{k+1} \leftarrow \eta_c b_{k+1}$
    $\Phi \leftarrow \eta_F \Phi$, $N \leftarrow \eta_F N$
  end if
end for

4-2 Experiments with Vanilla Actor-Critic

The performance of VAC is tested on an inverted pendulum swing-up task. This task is a standard benchmark in reinforcement learning (Buşoniu et al., 2010). First, the setup and the agent are introduced. Thereafter the simulation results are discussed. A separate section is devoted tot the influence of the forgetting factor η_F. This section ends with the results on the physical setup.

4-2-1 The Setup

The goal is to swing up a pendulum from a pointing down position to the upright position as quickly as possible and keep the pendulum upright by applying a voltage to the DC-motor. The action space is limited to $a \in [-3, 3]$ V, which makes it impossible to steer the pendulum directly from the start state to its upright position. Instead, the agent needs to pump energy in the system by swinging the pendulum back an forth until a swing to the top can be made. A picture of the setup is given in Figure 4-1. The dynamics of the pendulum are given by:

$$\begin{bmatrix}\dot{s}_1 \\ \dot{s}_2\end{bmatrix} = \begin{bmatrix}s_2 \\ \frac{1}{J}\left(Mgl\sin(s_1) - \left(z + \frac{K^2}{R}s_2\right)\right)\end{bmatrix} + \begin{bmatrix}0 \\ \frac{K}{JR}\end{bmatrix} a, \tag{4-11}$$

where s_1 is the angle of the pendulum and s_2 the angular velocity. The model parameters are stated in Table 4-1. The reward is calculated with a quadratic reward function

$$r_k(s_k, a_{k-1}) = -s_k^T Q s_k - a_{k-1}^T P a_{k-1} \tag{4-12}$$

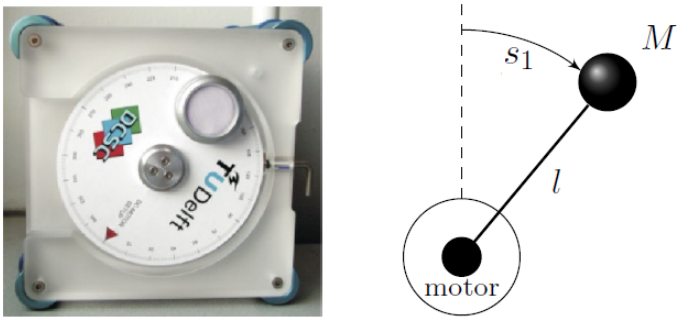


Figure 4-1: The inverted pendulum

Table 4-1: Parameters of the inverted pendulum

Model parameter	**Symbol**	**Value**	**Units**
pendulum inertia	J	$1.91 \cdot 10^{-4}$	kgm^2
pendulum mass	M	$5.50 \cdot 10^{-2}$	kg
gravity	g	9.81	m/s^2
pendulum length	l	$4.20 \cdot 10^{-2}$	m
damping	z	$3.0 \cdot 10^{-6}$	Nms
torque constant	K	$5.36 \cdot 10^{-2}$	Nm/A
rotor resistance	R	9.50	Ω
RL parameter	**Symbol**	**Value**	**Units**
sampling time	T_s	0.03	s
discount factor	γ	0.97	
exploration noise	σ^2	1	V
maximum action	$a_{\max}$	3	V
minumum action	$a_{\min}$	-3	V

Table 4-2: Parameters of equidistant grid of the RBFs for the inverted pendulum.

Variable		domain of centers	No BFs
angle(rad)	s_1	$[-\pi, \pi]$	10
angular velocity	s_2	$[-25, 25]$	10

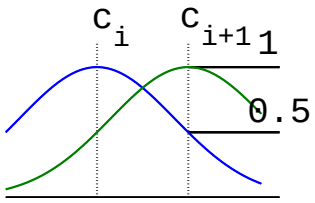


Figure 4-2: Choice of the scaling matrix B^{-1}.

with

$$Q = \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix} \qquad P = 1$$

The learning procedure is split into trials of 3 seconds after which the pendulum is reset to its downward position ($[\pi \quad 0]^T$). A sampling time of 0.03 s is used.

4-2-2 The Agent

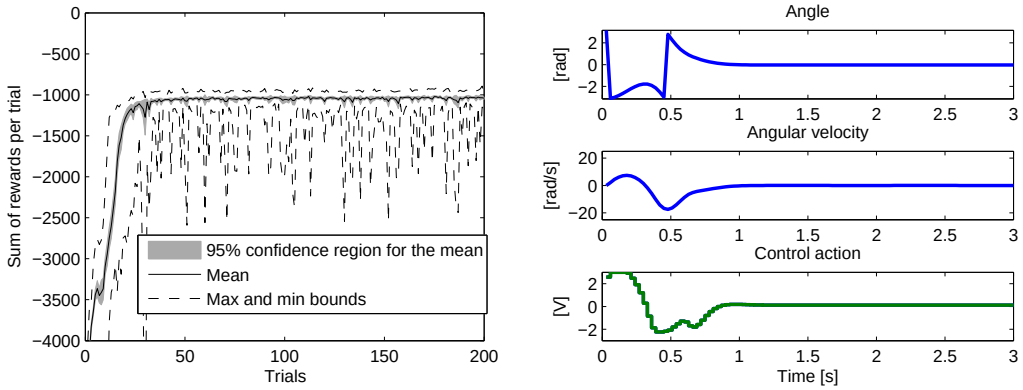
The actor selects actions using a Gaussian policy with linearly approximated mean and the standard deviation $\sigma = 1$. The critic also uses an approximator that is linear in the parameters. Actor and critic share the same set of basis functions, normalized Gaussian RBFs.

$$\phi_i(s) = \frac{\bar{\phi}_i(s)}{\sum_{i'=1}^{p} \bar{\phi}_{i'}(s)} \qquad \bar{\phi}_i(s) = \exp\left(-\frac{1}{2}(s - c_i)^T B_i^{-1}(s - c_i)\right) \tag{4-13}$$

The RBFs are placed on an equidistant grid. The parameters of the grid are stated in Table 4-2. The scaling matrix B is chosen such that the non-normalized Gaussian is half its maximum height at all its neighbors' centers. This is depicted in Table 4-2. Both the actor and the critic parameter vector are initialized at 0. The intial LSTD parameters are $A_0 = 10^{-3}I$, $b_0 = 0$. The actor is updated when the angle between consecutive natural gradients is less than 1°. The natural gradient is used because this is a better indication that the critic has converged. The natural gradient is only determined by the critic, where the vanilla gradient is also determined by the FIM.

4-2-3 Simulation Results

The algorithm and the MDP were implemented in `MATLAB`. A list of all the functions used is included in Appendix B. A single learning experiment lasts 200 trials, which for a trial length of 3 seconds is 10 minutes of system interaction. Learning is evaluated by comparing the average return received over 48 learning experiments.

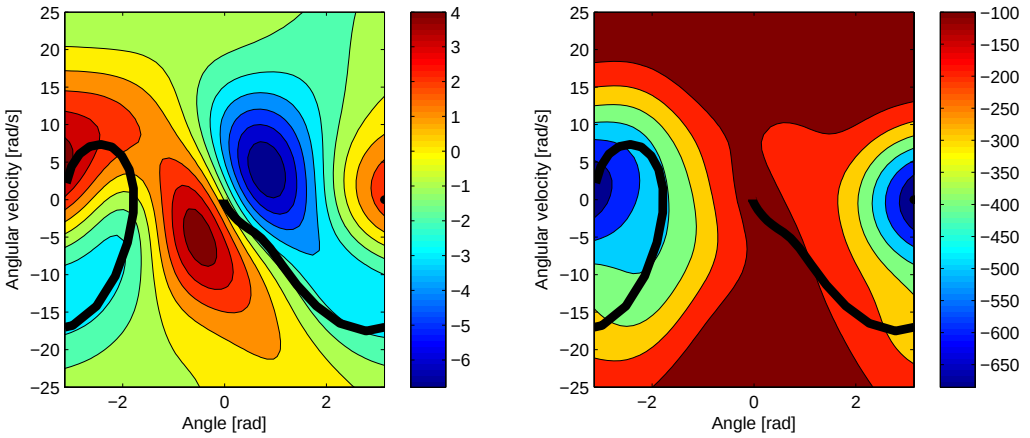


(a) Achieved learning behavior. **(b)** Typical trajectory.

Figure 4-3: The learning behavior of VAC on the simulated inverted pendulum, with the average received return (left) and a learned swing-up (right).

The parameters that need to be initialized are the constant actor learning rate α_a, the forgetting factor for LSTD-Q η_c, the forgetting factor for the FIM η_F and the eligibility trace factor λ. The parameters are determined by extensive tuning. The parameter sets are judged on two criteria. First, the algorithm must acquire the highest return possible. Since a number of different parameter setting converge to the same solution, the learning speed is the other criterion. The best set of parameters is $\eta_F = 0.9$, $\lambda = 0.6$, $\eta_c = 0.99$ and $\alpha_a = 0.3$. The achieved learning behavior and learned swing-up is given in Figure 4-3. The algorithm learns very quickly that it can get the pendulum higher by using swings. The agent learns steadily how to pump more energy in the system. After approximately 6 trials the pendulum reaches the top for the first time. The trial thereafter the agent adds more energy. This causes the pendulum to overshoot the top and swing a couple of times around. This causes the little kink after 6 trials in Figure 4-3a. Then, the agent learns that by adding slightly less energy with the swings, the pendulum stays upright longer. At the same time the agent discovers that the pendulum can be stabilized at the top. This results in the rapid increase in the return in Figure 4-3a. From approximately 17 trials a steady swing-up has been learned. Thereafter the agent learns to perfect the swing-up stabilizing controller. The algorithm converges after 40 trials, corresponding to 2 minutes of system interaction. The spikes in the min bounds are caused by occasional failed swing-ups. These are caused by the exploration noise in the policy. The final learned swing-up without exploration noise is shown in Figure 4-3b. The swing-up action takes about 1 second. After that point the pendulum is at rest at a very slight angle. With exploration noise the offset could not be found in the data. Because the optimal control action at the top position ($[0 \quad 0]^T$) is zero, the exploration noise is the dominant part of the applied control action. The learned policy and value function are given in Figure 4-4. The desired control action is larger than allowed for parts of the state space. This has two causes:

- The agent circumvents the exploration noise. In crucial parts of the state space where the maximum control action is required it is better for the agent to select an action that is higher than the saturation boundary. The agent then has a bigger chance that the action with exploration noise is still the maximum action. The additional reward paid for the bigger desired action is compensated for by steadier behavior. This is the case



(a) Policy. **(b)** Value function.

Figure 4-4: The learned actor and critic by VAC on simulations of the inverted pendulum. The black line is a learned swing-up.

in the early phase of the swing up at $[-\pi \quad 4]^T$.

- The policy is optimized for the trajectory that is followed most often. The quality of the policy of other parts of the state space is "sacrificed" to get the best performance in the operating band optimal.

4-2-4 The Influence of the Forgetting Factor

For VAC algorithm introduces and additional learning parameter that needs to be set: the FIM forgeting factor η_F. The influence of the FIM forgetting factor η_F on the learning behavior is discussed in this section. The system is simulated using multiple values for η_F. The other agent parameters (the constant actor learning rate α_a, the forgetting factor for LSTD-Q η_c and the eligibility trace factor λ) are tuned for maximum performance. The best values are stated in Table 4-3. The influence of the forgetting factor η_F is shown in Figure 4-5, where the episodic return is shown, averaged over 48 learning experiments. The 95% confidence bounds of the separate experiments are shown in Figure 4-6. The amount of information in the FIM is determined by the length between consecutive updates and the forgetting factor η_F. With LSTD-Q the natural gradient w converges in approximately 2-5 time steps. On average between 20 and 50 updates are performed per trial. For values of η_F that yield the fastest learning ($\eta_F \leq 0.95$), most information added at the beginning of the trial is lost at the end. The estimate of the vanilla gradient obtained with F does not have to be close to the actual vanilla gradient, because the steady state distribution for small N and η_F is not properly captured in F.
With an actor using RBFs and short intervals between actor updates another phenomenon occurs. For a feature vector $\phi_{\mathrm{a}}(s_k)$, most elements $\phi_{\mathrm{a},i}(s_k)$ are approximately zero. Therefore most elements of $\phi_{\mathrm{a}}(s_k)\phi_{\mathrm{a}}(s_k)^T$ will be approximately zero. For small N and a small forgetting

Table 4-3: Agent parameters, tuned per forgetting factor η_F.

η_F	λ	η_c	α_a
0	0.6	0.99	0.2
0.5	0.6	0.99	0.3
0.9	0.6	0.99	0.3
0.95	0.8	0.99	0.2
0.999	0.6	0.99	0.1
1	0.8	0.90	0.2

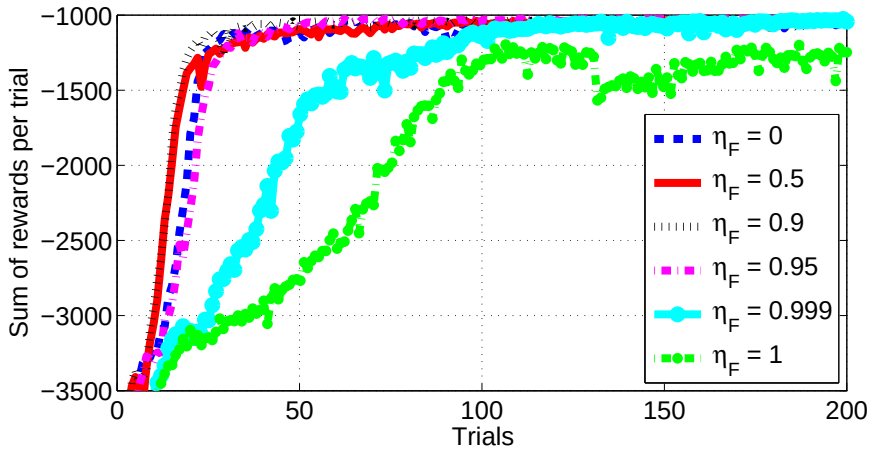


Figure 4-5: Influence of the FIM forgetting factor η_F on the learning behavior of VAC on the simulated inverted pendulum.

factor η_F the matrix Φ will be sparse. The actor update Equation (4-9) can be rewritten as:

$$\vartheta_{k+1} \approx \vartheta_k + \frac{\alpha}{N\sigma^2} \sum_{i=0}^{N} \phi_{\mathrm{a}}(s_i) u_i, \tag{4-14}$$

$$u_i = \phi_{\mathrm{a}}(s_i)^T w. \tag{4-15}$$

The entries of the actor parameter vector $\vartheta(j)$ are updated according to the output of the corresponding BF $\phi_{\mathrm{a},j}(s)$, that is $\Delta\vartheta(j) \approx \frac{\alpha}{N\sigma^2} \sum_{i=0}^{N} \phi_{\mathrm{a},i} u_i$. For small η_F and N only entries of the policy vector $\vartheta(i)$ will be significantly changed if the corresponding BFs $\phi_{\mathrm{a},i}$ has been significantly bigger than zero in the update interval. In each update only a small part of the policy vector ϑ will be updated. A discussion on the differences between the vanilla and the natural gradient update is given in Section 4-3.

4-2-5 Experimental Results

Because the simulation results of VAC are very promising, VAC is tested on a physical setup. The algorithm is rewritten to communicate with the real time system. The following adaptations are made:

- A reset mechanism is added to steer pendulum to its downward position as quickly as possible if the trial has finished.

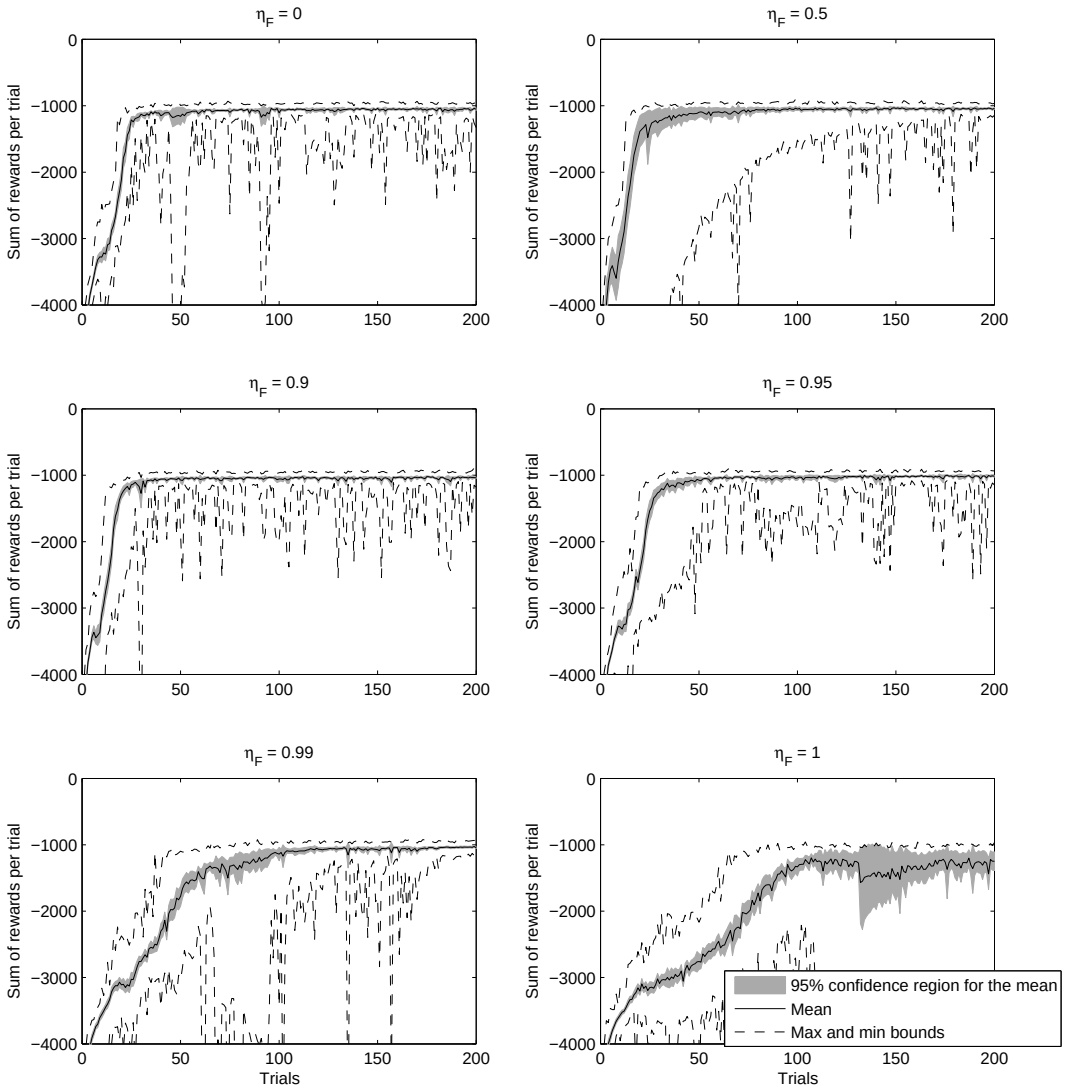


Figure 4-6: Learning behavior of VAC on the simulated inverted pendulum for different values of the FIM forgetting factor η_F.

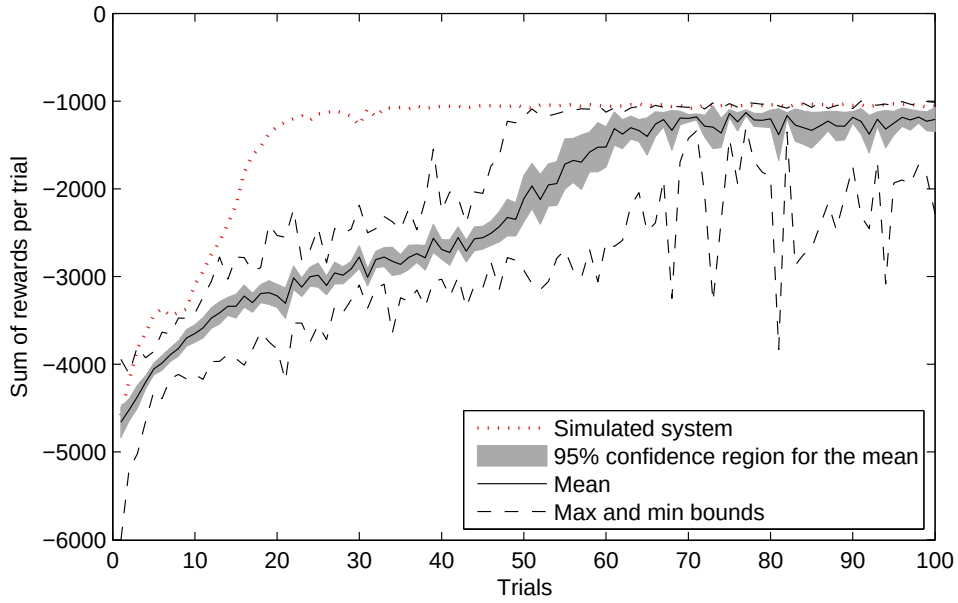


Figure 4-7: Cumulative reward of VAC on the setup (black solid) and in simulated (red dotted).

- The model (Equation (6-37)) is used as an observer for two reasons
  - The setup does not have an accurate angular velocity sensor. A finite difference estimate of the angular velocity is not accurate. Estimating the angular velocity with the nonlinear model proved to work best. The inputs for the model are the measured state and a finite difference angular velocity estimate. The agent receives the measured angle and the modeled angular velocity.
  - Occasionally large sensor errors occur. These errors disturb the learning behavior because the critic is updated with faulty state information. This has shown to degrade the performance. When there is a too big discrepancy between the measured state and the observer the trial is aborted and restarted. Also all parameters are reset to their value at the start of the failed trial.

With the modifications the algorithm is tested with the optimal simulation parameters on the pendulum setup. The average episodic reward of 20 learning procedures is shown in Figure 4-7 along with the learning behavior during simulation. There are three main differences between the simulation and the experiment. First, the learning speed is much lower on the setup. The algorithm needs approximately three minutes to converge. In simulation a swing-up is learned in a third of the time needed on the true system. Second, the shape of the learning curve is different. The tiny dip in the simulated learning is not present in the learning process on the true system. It takes VAC longer to stabilize the pendulum at its top position on the setup. This is the phase between trial forty and sixty. Third, the final performance is lower on the setup than in the simulation. The main causes of the decreased performance are:

- State estimation. The angle sensor sometimes gives faulty readings. Although the biggest mistakes are rejected by restarting the trial, faulty state measurements lead to

a wrong estimation of the critic: both the transition and the accompanying reward are wrongly calculated. The mistakes in the critic cause bad actor updates which in turn lead to slower learning.

- Model complexity. The model of the pendulum is very simplistic. Non-modeled dynamics, such as nonlinear friction decrease the performance. To overcome the non-modeled friction forces a higher control action is required. Also the model parameters could be slightly off. This is probably the cause of the slightly lower return of the final policy.
- Time delays. These are introduced by the digital control and the required computation time.

4-3 Differences with the Natural Gradient

The influence of the type of gradient used is investigated in this section. The performance of VAC and Natural Actor-Critic (NAC) is first compared in simulation and thereafter on the setup. NAC is implemented in `MATLAB`. Two modifications were made compared to Algorithm 2.1:

- After an actor update the critic has to forget its statistics. When the A-matrix of LSTD is discounted, the fraction of the initial matrix that is lost due to the discounting is added[1]:
$$A_{k+1} \leftarrow \eta_c A_{k+1} + (1-\eta_c)A_0. \tag{4-16}$$
- The natural gradient has to be normalized in order to guarantee stable learning on the inverted pendulum. The policy update equation is:
$$\vartheta_{k+1} = \vartheta + \alpha_a \frac{w}{\|w\|}. \tag{4-17}$$

The initial agent parameters were set to $\vartheta_0 = 0$, $\theta_0 = 0$. The other learning parameters are $\alpha_a = 0.3$, $\lambda = 0.85$, $\eta_c = 0.95$. These are also stated in Table 4-4.
Simulations of the inverted pendulum showed that without the normalization the only way divergence can be prevented is choosing learning rates in the order of 10^{-6}. This leads to undesirable slow learning. Although not mentioned in (Peters and Schaal, 2008a), the episodic NAC in the toolbox (Peters, 2002) also uses a normalization before returning the natural gradient. The bad natural gradient update can be caused by two main reasons:

- Mistakes in the estimate of w can accumulate in the actor update. The parametrization is such that the entry $w(i)$ is updated according to:[2]
$$w(i)_{k+1} = w(i)_k + \alpha_c \delta_{k+1} \frac{a_k - \mu(s_k)}{\sigma^2} \phi_{a,i}(s_k). \tag{4-18}$$
Because the value of $w(i)$ is only changed if $\phi_{a,i}(s)$ is nonzero, $w(i)$ can have the same value for an extensive period of time. For instance if the generalization is not very

[1] This is also done in VAC. In Section 4-1 a more thorough explanation is given.
[2] The TD-update is stated here, because it provides an easier insight into this phenomenon

Table 4-4: Parameters ofVAC, TD-VAC, NAC, TD-NAC and unnormalized TD-NAC for simulations of the inverted pendulum swing-up.

Parameter		NAC	TD-NAC	Normalized TD-NAC	VAC	TD-VAC
actor learning rate	α_a	0.3	0.05	0.02	0.3	0.05
critic learning rate	α_c		0.4	0.4		0.1
trace decay rate	λ	0.85	0.85	0.8	0.6	0.85
critic forgetting factor	η_c	0.95	0	0.95	0.99	1
FIM forgetting factor	η_F				0.9	0.9

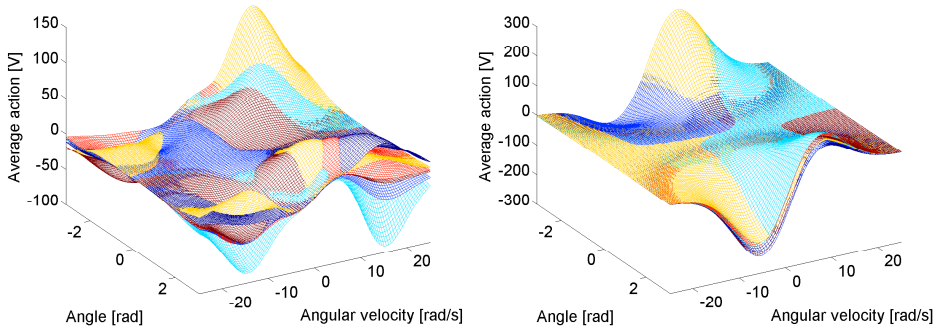
broad, RBFs will have a value that is approximately zero if the state s is not close to the center position:

$$\phi_{\mathrm{a},i}(s) \approx 0 \qquad \text{if} \quad \|c_i - s\| > \Delta. \tag{4-19}$$

where c_i is the center of the i-th RBF and Δ is a small distance . When during learning a single trajectory comes close to the center of the RBF, such that $\|c_i - s\| < \Delta$, $\phi_{a,i}$ will be nonzero during that trajectory. This can set $w(i)$ to be nonzero. If the following trajectories do not again come near c_i, the BF will from there on be $\phi_{a,i} \approx 0$. Therefore $w(i)$ will stay approximately constant. This leads to actor divergence, because every update $\vartheta(i)$ is changed with $\alpha_a w(i)$.

- The LSTD overfits the parameter vector. For RBF's entries of the parameter vector are changed that correspond to BFs with centers that lie in unvisited parts of the state space.

The first probable cause is tested by setting the forgetting factor $\eta_c = 0$. For this setting NAC still diverges. The second potential problem is tested by replacing the LSTD critic by a TD critic. TD-NAC is able to learn a proper policy when the advantage function parameter w is completely forgotten after each update. Otherwise the magnitude of the natural gradient increases, leading to divergence. This shows that both identified causes have a big influence on the actor update.
The vanilla algorithm does not suffer from these phenomena because the multiplication with the incomplete Fisher matrix removes these influences. This is demonstrated in the following experiment: during one trial for the zero policy $\vartheta = 0$ during one trial the natural and the vanilla gradient are learned using LSTD-Q. The difference between different vanilla gradient estimates is much smaller than the difference between the natural gradients estimates. The standard deviation of the angle between the different natural gradients is 13.3°, compared to only 2.7° for the vanilla gradient. This is visualized in Figure 4-8. The desired policy update is plotted for six gradient estimates for the natural gradient (left) and the vanilla gradient obtained from these natural gradients (right). The natural gradient estimates defer both in size and in shape. There are peaks present in one update that are not present in other updates. The updates with the vanilla gradient are more much more similar in shape and in size
To investigate the influence of the critic and the gradient type used on the learning process, VAC, NAC, TD-VAC and TD-NAC are tested on a simulation of the inverted pendulum. The parameters were found by extensive tuning. The best parameters are stated in Table 4-4.
The average learning behavior during 48 learning experiments is shown in Figure 4-9. Based on these results the following conclusions can be drawn:



(a) Natural gradient update. **(b)** Vanilla gradient update.

Figure 4-8: Actor update on the inverted pendulum for the zero policy $\vartheta = 0$.

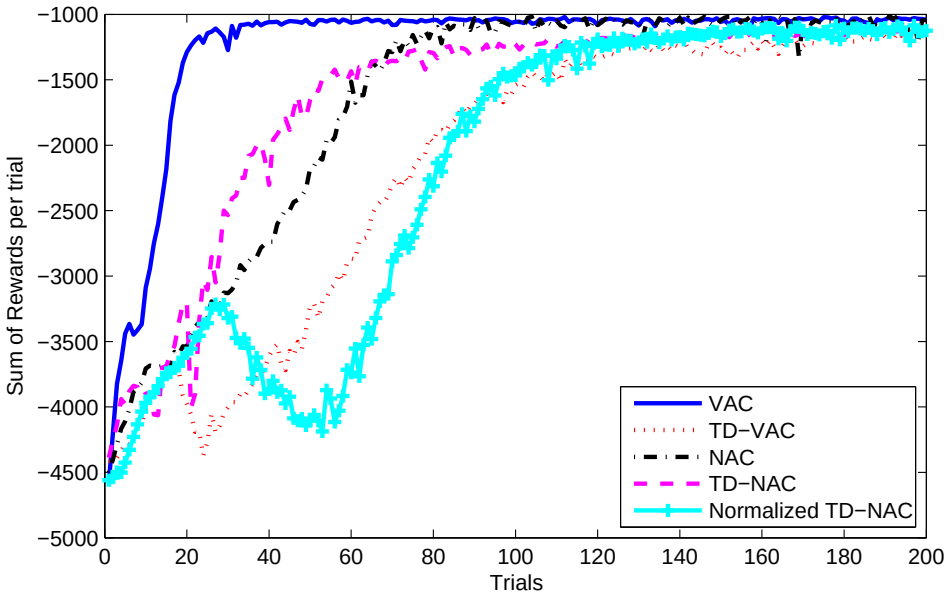
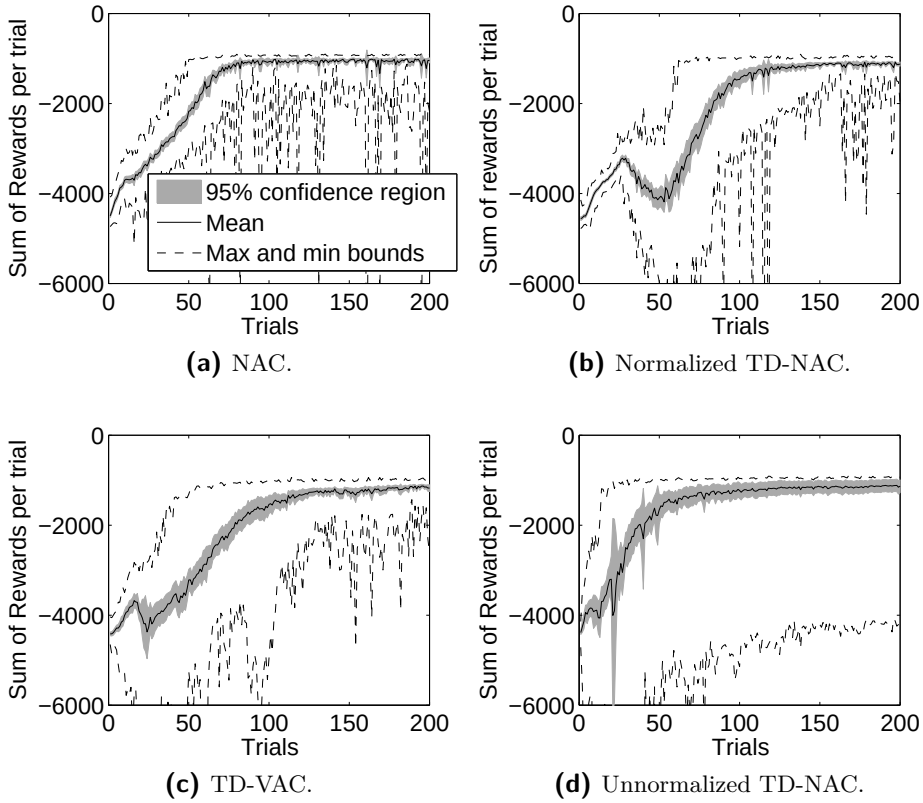


Figure 4-9: Learning behavior of VAC, TD-VAC, NAC, TD-NAC and unnormalized TD-NAC on simulations of the inverted pendulum.



(a) NAC. **(b)** Normalized TD-NAC.

(c) TD-VAC. **(d)** Unnormalized TD-NAC.

Figure 4-10: Confidence bounds of the mean of the learning behavior of NAC, normalized TD-NAC, TD-VAC and unnormalized TD-NAC on simulations of the inverted pendulum swing-up task.

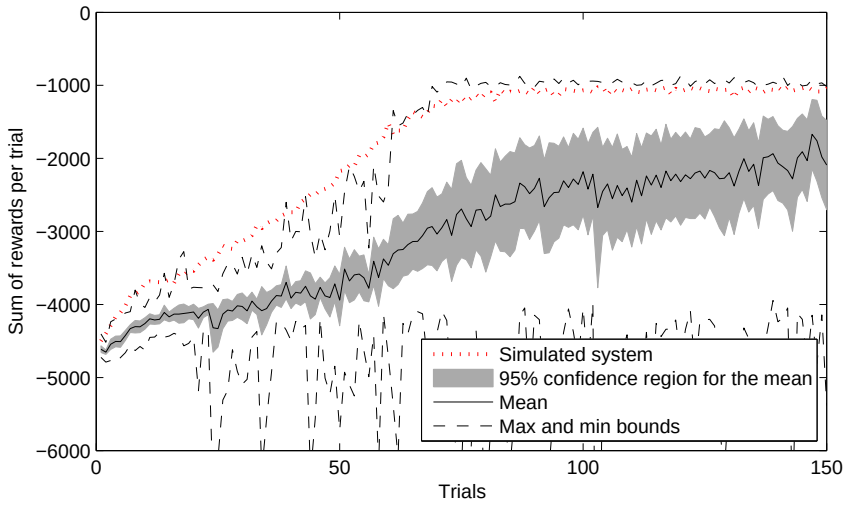


Figure 4-11: Cumulative reward of NAC on the pendulum setup.

- The normalization prevents divergence but also slows down the learning because the size of the gradient is not exploited.

- Algorithms with a TD critic update generally learn more slowly than algorithms with an LSTD critic for the same actor update. VAC-LSTD and normalized NAC-LSTD learn faster than their TD-versions. A similar result was observed for standard-actor critic (Section 2-2-3) and VGBP (Section 6). But care has to be taken, because LSTD is susceptible to overfitting, which can lead to instability. This instability is related to the parameterization of the agent. If in a parametrization certain features $\phi_{c,i}(s)$ have a very small output compared to other features the least-squares optimization can assign a very large critic parameter $\theta(i)$ to the feature to minimize the overall TD-error.

- For the TD-update the use of the natural gradient is beneficial. It is assumed that, if by a different parameterization no normalization is needed, NAC will outperform VAC. In VAC the features $\phi_i(s)$ that are susceptible to overfitting are not used in the update of the policy vector. Because NAC uses the natural gradient directly it is more susceptible to overfitting by the LSTD critic.

4-3-1 Natural Actor-Critic on the Setup

The code of NAC is rewritten with the same modifications as VAC to learn the swing-up on the setup. The results of 20 learning experiments are shown in Figure 4-11. The performance of NAC suffers more from the transition to the physical system than VAC. There are more fluctuations in the return during a learning experiment on the setup than in simulation. NAC does not succeed in finding a swing-up for all experiments within 7.5 minutes of system interaction. The measurement noise is the cause that the learned natural gradient is of bad quality.

4-4 Conclusion

In this chapter a vanilla actor-critic method is devised that uses a Gaussian policy. The vanilla gradient is estimated with an approximation of the FIM and the natural gradient. For the inverted pendulum swing-up task this algorithm has shown to outperform natural gradient actor-critic algorithms. The main reason that this algorithm performs better is that the estimate of the FIM removes the effects of overfitting of the critic in the actor update. The performance of both VAC and NAC is much better in simulation than when using a physical setup.

Chapter 5

The Policy Hessian

The second important ingredient of the Levenberg Marquardt (LM) algorithm is the Hessian. Therefore an LM actor update needs the policy Hessian. In principle LM makes use of the true Hessian, but it is not always possible or convenient to use the true Hessian: the second order derivative of the objective function may be unknown, or too computationally expensive to compute. In general less accuracy is required in the Hessian than in the gradient estimate (Byrd et al., 2011). The eigenvalues of the Hessian are used to determine the quality of the Hessian approximation, because:

- The Hessian estimate needs to be negative definite near an optimum to fully exploit the LM.
- The largest eigenvalue of the Hessian is the optimal step-size for a quadratic objective function (Lecun et al., 1993). If the full Hessian estimate leads to bad updates, but the largest eigenvalue can be properly estimated, it can be used as learning rate.

In this chapter first an analytical expression is sought for the policy Hessian. Thereafter Hessian approximation techniques are used to find the policy Hessian.

5-1 Mathematical Derivation

To derive a formula for the Hessian, the proof of the Policy Gradient Theorem in Sutton et al. (1999) can be extended by using the second derivative w.r.t. ϑ. In Appendix A-2 it is proven that for the average reward setting the Hessian is:

$$\frac{\partial^2 \rho}{\partial \vartheta^2} = \sum_s d^\pi(s) \sum_a \left[\frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} Q^\pi(s,a) + \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial Q^\pi(s,a)}{\partial \vartheta}^T + \frac{\partial Q^\pi(s,a)}{\partial \vartheta} \frac{\partial \pi(s,a)}{\partial \vartheta}^T \right]. \quad (5\text{-}1)$$

The same result was established in a different way in Kakade (2001b). For a Gaussian policy with a linearly approximated mean, the Hessian is:

$$\frac{\partial^2 \rho}{\partial \vartheta^2} = F \frac{\partial w}{\partial \vartheta} + \frac{\partial w}{\partial \vartheta}^T F, \quad (5\text{-}2)$$

Table 5-1: Quality of the policy Hessian and gradient estimates made with HMDP on the simulated inverted pendulum.

	Largest eigenvalue of H		Standard deviation gradient estimate
ϑ	mean	standard deviation	
ϑ^*	-22	89	26°
0	-23	86	21°

where F is the Fisher Information Matrix (FIM) and w is the natural gradient. The complete derivation is given in Appendix A-1-4. Although Equation (5-2) shows that the Hessian can be obtained using a similar transformation as used in forming the vanilla gradient from the natural gradient, it does not provide any insight into how the Jacobian of the advantage function parameter should be estimated.

5-1-1 Hessian for a Markov Decision Process

In Kakade (2001b) an algorithm is given that estimates the Hessian (A-18), called Hessian for Markov Decision Process (HMDP). The method is stated in Algorithm 5.1. The proof that the algorithm provides an unbiased estimator of the Hessian is quite long, hence the interested reader is referred to Kakade (2001b).

Algorithm 5.1 HMDP (Hessian for a Markov Decision Process)

initialize state s_0, $z_0 = g_0 = 0 \in \mathbb{R}^n$, $\tilde{z}_0 = y_0 = H_0 = 0 \in \mathbb{R}^{n \times n}$
for every step $k = 1, 2, \ldots$ **do**
  draw action $a \sim \pi$
  observe next state s_{k+1} and reward r_{k+1}
  $y_{k+1} = \gamma \left(y_k + \frac{\nabla \pi(s,a)}{\pi(s,a)} z_k^T + z_k \frac{\nabla \pi(s,a)}{\pi(s,a)}^T \right)$
  $z_{k+1} = \gamma \left(z_k + \frac{\nabla \pi(s,a)}{\pi(s,a)} \right)$
  $\tilde{z}_{k+1} = \gamma \left(\tilde{z}_k + \frac{\nabla^2 \pi(s,a)}{\pi(s,a)} \right)$
  $g_{k+1} = g_k + r_{k+1} z_{k+1}$
  $H_{k+1} = H_k + r_{k+1} \tilde{z}_{k+1} + r_{k+1} y_{k+1}$
end for
gradient $g = g_k / k$
Hessian $H = H_k / k$

To see if HMDP works well in practice it is tested on the inverted pendulum. For a fixed policy the Hessian and the gradient are estimated in a single trial. The Hessian estimates of 500 trials are compared by their eigenvalues. The experiment and agent parameters are the same as used in Section 4-2. The agent's behavior is described by a Gaussian policy that is linear in the parameters (LIP). The standard deviation of the exploration noise is set to $\sigma = 1$. This experiment is conducted with two policies: the zero policy with $\vartheta = 0$ and an estimate of ϑ^* learned using VAC. The results are stated in Table 5-1. The experiments show

that, although HMDP converges theoretically, it is not properly implementable in an LM actor update, because:

1. The Hessian estimate is not negative definite near the optimum.
2. The extremely high variance of the largest eigenvalue makes it not usable as a step size. The largest eigenvalue can not be estimated efficiently enough to make up for the additional samples needed to determine a better step size.

The variance in the gradients estimates pf HMDP is much bigger than the variance in the gradient estimates of VAC and NAC (2.7◦ and 13.3°).

5-1-2 Gauss-Newton

For (nonlinear) least squares problems the Hessian can be approximated by

$$\tilde{H}(x) = J(x)^T J(x), \tag{5-3}$$

where $J(x)$ is the Jacobian of the objective function at x. In order for $\tilde{H} \in \mathbb{R}^{n \times n}$ to be invertible, the Jacobian requires a column rank of at least n. For instance when $f(x)$ resembles a least squares problem, at least n data points are needed to obtain a full rank Hessian estimate. The approximated Hessian is valid in the neighborhood of the optimum where the gradient and the second order derivative are weakly correlated (Verhaegen and Verdult, 2007) and is always positive definite. In (Chen, 2011) it is shown that the Gauss-Newton approximation is preferred to the Hessian in the presence of noise for both the Newton and the LM method. The Gauss-Newton method can not be used in the actor update because the policy gradient has rank 1 and the approximation is always positive definite.

5-1-3 An Approximate Newton Method

In Furmston and Barber (2012) a method is devised to estimate the Hessian for an MDP framework with a finite planning horizon and discrete state and action spaces. For this finite horizon K the return is given by:

$$\rho(\pi) = \sum_{k=0}^{K} \sum_{\mathcal{S}} d_k^{\pi}(s) \sum_{\mathcal{A}} \pi_{\vartheta,k}(s,a) \mathcal{R}(s,a), \tag{5-4}$$

where $d_k^{\pi}(s)$ is the marginal $p(s_k = s \mid \pi, s_0)$ and $\pi_k(s,a) = p(a_k = a \mid s_k = s)$. The return can also be written as the result of all trajectories $(s_0, a_0, \ldots, s_K, a_K)$. Let the set of all trajectories with length K be denoted by $\mathcal{T}_K$. The probability that a trajectory $p(s_{1:K}, a_{1:K})$ is followed is:

$$p(s_{1:K}, a_{1:K}) = \pi(s_K, a_K) \left\{ \prod_{k=0}^{K-1} \mathcal{P}(s_k, a_k, s_{k+1}) \pi(s_k, a_k) \right\} p(s_0). \tag{5-5}$$

The return derived using trajectories is:

$$\rho(\pi) = \sum_{k=0}^{K} \sum_{\mathcal{T}_k} p(s_{1:k}, a_{1:k}) \mathcal{R}(s_k, a_k). \tag{5-6}$$

The policy gradient is found by using the log-trick $\frac{\partial}{\partial\vartheta}p = p\frac{\partial}{\partial\vartheta}\log p$. With this log-trick the derivative of Equation (5-6) is:

$$\frac{\partial\rho}{\partial\vartheta} = \sum_{k=0}^{K}\sum_{\mathcal{T}_k} p(s_{1:k}, a_{1:k})\mathcal{R}(s_k, a_k)\frac{\partial \log p(s_{1:k}, a_{1:k})}{\partial\vartheta}. \tag{5-7}$$

One possible way to solve Equation 5-7 is by using the Policy Gradient Theorem:

$$\frac{\partial\rho}{\partial\vartheta} = \sum_{k=0}^{K}\sum_{\mathcal{S}} d_k^{\pi}(s)\sum_{\mathcal{A}}\pi_{\vartheta,k}(s,a)Q_k(s,a)\frac{\partial \log \pi_{\vartheta,k}(s,a)}{\partial\vartheta}, \tag{5-8}$$

where $Q_k(s,a)$ is the expected return received from time step k until K after taking a_k in s_k. The Hessian of the return can be derived by differentiating Equation 5-7, and again using the log-trick:

$$H(\vartheta) = \sum_{k=0}^{K}\sum_{\mathcal{T}_k} p(s_{1:k}, a_{1:k})\mathcal{R}(s_k, a_k)\left[\frac{\partial \log p(s_{0:k}, a_{0:k})}{\partial\vartheta}\frac{\partial \log p(s_{0:k}, a_{0:k})}{\partial\vartheta}^T + \frac{\partial^2 \log p(s_{0:k}, a_{0:k})}{\partial\vartheta^2}\right] \tag{5-9}$$

$$= H_1 + H_2. \tag{5-10}$$

Because a negative definite Hessian is needed for the Newton method and the matrix H in general may not be negative definite, Furmston and Barber (2012) use a Hessian approximation that is always negative definite. Possible candidates are H_1 and H_2. The matrix H_1 is an outer product and thus always positive-semidefinite. Although the matrix H_2 does not have to be negative definite, for certain policies negative definiteness can be proven. For a Gaussian policy with fixed standard deviation σ the matrix H_2 is negative definite over the mean parameter space (Furmston and Barber, 2012). Therefore H_2 will be further used as a Hessian approximation. The derivative of the logarithm of the trajectory probability of H_2 in Equation (5-9) can be simplified, by noticing that $\log p(s_{1:k}, a_{1:k}) = \sum_{k=0}^{K}\log\pi(s_k, a_k) + \sum_{k=0}^{K}\log\mathcal{P}(s_k, a_k, s_{k+1})$. The last part drops out of the equation because its derivative with respect to ϑ is zero. With this insight the Hessian estimate H_2 can be rewritten as:

$$H_2 = \sum_{k=0}^{K}\sum_{\mathcal{T}_k} p(s_{1:k}, a_{1:k})\mathcal{R}(s_k, a_k)\left[\sum_{\kappa=0}^{k}\frac{\partial^2 \log\pi(s,a))}{\partial\vartheta^2}\right] \tag{5-11}$$

$$= \sum_{k=0}^{K}\sum_{\mathcal{S}} d_k^{\pi}(s)\sum_{\mathcal{A}}\pi_{\vartheta,k}(s,a)Q_k(s,a)\left[\frac{\partial^2 \log\pi(s,a))}{\partial\vartheta^2}\right]. \tag{5-12}$$

An unbiased estimate of Equation (5-12) is found by sampling according to $\pi_\vartheta(s,a)$. An unresolved question is if this result can be extended to the infinite horizon setting. Due to time constraints[1] this extension has not been proven. It is assumed that a valid estimator in the infinite time setting may have the form of:

$$H_2(\vartheta) = -\int_{\mathcal{S}} d^{\pi}(s)\int_{\mathcal{A}}\pi_\vartheta(s,a)Q(s,a)\left[\frac{\partial^2 \log\pi_\vartheta(s,a)}{\partial\vartheta^2}\right]\mathrm{d}a\mathrm{d}s. \tag{5-13}$$

[1]The paper was found two weeks before the deadline of the project.

Personal communication with Furmston showed that the authors also have this assumption. It is strengthened by their successful implementation to a Tetris MDP. Unfortunately no proof has been derived.[2] Although no proof has been found, a short speculation about the consequences of this Hessian approximation is given. In Appendix A-3 it is proven that when Equation (5-13) is valid, the Hessian approximation H_2 for a Gaussian policy that is LIP is given by:

$$H_2 = -\frac{1}{\sigma^2}\int_{\mathcal{S}} d^\pi(s)b(s)\phi_{\mathrm{a}}(s)\phi_{\mathrm{a}}(s)^T \mathrm{d}s, \tag{5-14}$$

where $b(s)$ is a baseline function. To guarantee negative definiteness $b(s)$ needs to be non-negative. In this thesis the value function $V(s)$ is used as the baseline. To get a positive value function all rewards need to be nonnegative. For a bounded reward function this can be easily ensured. An approximation of the Hessian then becomes:

$$H_2 \approx -\frac{1}{N\sigma^2}\sum_{i=1}^{N} V(s)\phi_{\mathrm{a}}(s)\phi_{\mathrm{a}}(s)^T. \tag{5-15}$$

This matrix has the following interesting properties:

- It has a structure very similar to the FIM.
- The matrix H_2 is influenced by the base line $b(s)$ and not by the natural gradient w.
- It is an outer product. For the matrix $H_2 \in \mathbb{R}^{n\times n}$ to be invertible at least n samples are needed and all BFs $\phi_{\mathrm{a}i}(s)$ need to have been nonzero at least once.

The last point shows the difficulty of this method. For instance for the inverted pendulum for a fixed policy large parts of the state space are unvisited for an extended period of time. Therefore the feature values of RBF's with centers in the unvisited areas will be approximately zero. This Hessian approximation will almost never be properly scaled. To see if the assumption is valid the method is tested in `MATLAB`.

Experiment

The quality of the Hessian approximation H_2 is tested on simulations of the inverted pendulum. VAC is used to generate the actions and learn the critic function. The policy is kept constant: $\vartheta = 0$. It is learned in a single trial of 4 seconds, which can be enough to obtain a full rank Hessian. The eigenvalues will be used to determine the quality of the Hessian.
To learn the Hessian the following adaptations were made:

- The reward is calculated with:

$$r_{k+1} = -s_{k+1}^T Q s_{k+1} - a^T P a + 150, \tag{5-16}$$

  where P and Q are defined in Section 4-2-1. The constant is added to guarantee that r_{k+1} is positive.

[2] This method has only been tested on actor-only algorithms. Thomas Furmston is writing a journal paper on actor-critic implementations.

- After each action selection the Hessian is updated with:

$$\mathcal{H}_{k+1} = \mathcal{H}_k + \tilde{V}(s)\phi_{\mathrm{a}}(s_k)\phi_{\mathrm{a}}(s_k)^T, \tag{5-17}$$

  where $\mathcal{H}_0 = 0$ and $\tilde{V}(s) = \max(V(s), 0)$. Due to the least-squares fit of the value function, some states may have a negative value. The maximization ensures that $\mathcal{H}$ is always negative definite.

- When the Hessian H_2 is needed, it is computed with:

$$H_2 = -\frac{1}{k}\mathcal{H}_k. \tag{5-18}$$

Simulations have shown that the eigenvalue with the largest magnitude is -412 and the eigenvalue with smallest magnitude is approximately zero. The condition of the matrix is very bad, making it useless for a Newton method. The estimate of the largest eigenvalue is very stable. Therefore the inverse of the largest eigenvalue might be used as an actor step size. Due to time constraints it has not been investigated if this step size works well in an episodic version of VAC.

5-2 Hessian Approximation

Finite difference approximations of the Hessian have been successfully applied to supervised learning (Becker and LeCun, 1989; Byrd et al., 2011; Spall, 2000), but to the author's best knowledge no papers have used finite difference methods to create an efficient second order actor update.

5-2-1 The Broydon-Fletcher-Goldfarb-Shanno Method

By evaluating the gradient of the objective function at a number of points around x, a finite difference estimate of the Hessian can be obtained. An often used method is the *Broydon-Fletcher-Goldfarb-Shanno* method (Luenberger and Ye, 2008):

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{q_{k+1}q_{k+1}^T}{q_{k+1}^T\Delta x_k} - \frac{\tilde{H}_k\Delta x_k\Delta x_k^T\tilde{H}_k}{\Delta x_k^T\tilde{H}_k\Delta x_k}, \tag{5-19}$$

where Δx is the step taken from x_k to x_{k+1} and q_{k+1} is defined as:

$$q_{k+1} = J(x_{k+1}) - J(x_k). \tag{5-20}$$

The initial Hessian can be chosen as $H_0 = I$.
To use BFGS to learn the policy Hessian, the step $\Delta\vartheta$ from ϑ_k to ϑ_{k+1} needs to be chosen. An obvious choice for $\Delta\vartheta$ is the vanilla gradient ascent step. The Hessian is then estimated during the regular learning process and when the Hessian estimate has converged it can be used to choose the step size. This concept is tested on the inverted pendulum. The gradient is learned with VAC. The standard model and agent parameters of Section 4-2 are used.
The quality of the Broydon-Fletcher-Goldfarb-Shanno (BFGS) Hessian is tested with the

Table 5-2: Largest eigenvalues of the Hessian $H(\vartheta)$ learned with BFGS.

	Vanilla gradient		Natural Gradient	
ϑ	Mean	Standard deviation	Mean	Standard deviation
ϑ^*	-6.43^3	9.89×10^4	-4.14×10^3	2.47×10^4
0	3.12×10^3	4.55×10^4	-8.44×10^3	5.13×10^4

following experiment: starting with an initial policy ϑ_0, the Hessian is learned during one trial. This is long enough to remove the influence of the initial Hessian H_0. This is short enough that it is assumed that the learned parameter vectors ϑ are close enough to their initial value, such that the approximated Hessian is close to the Hessian at ϑ_0. This experiment is repeated a hundred times for the initial parameter vector $\vartheta_0 = 0$ and $\vartheta_0 = \vartheta^*$. The optimal policy vector ϑ^* has been learned with regular VAC. The quality of the method is determined by evaluating the mean and the standard deviation of the largest eigenvalue of the Hessian. The mean and the standard deviation of the largest eigenvalue of the Hessian are stated in Table 5-2. For both initial policies the Hessian is in general not positive definite. The largest eigenvalue cannot be used as a step size due to the large variance (it even changes sign). The large variance is caused by the variance of the gradient estimates amplified by the small steps $\Delta\vartheta$. The steps $\Delta\vartheta$ can not be chosen larger, because then the finite difference approximation is not valid. The Hessian approximation is not negative definite, even for the optimum initial parameter. The same experiment is repeated with the natural gradient using NAC. The results are stated in Table 5-2. The Hessian estimate obtained with the natural gradient suffers from the same problems as the estimate acquired with the vanilla gradient. Because the policy Hessian is not properly estimated with BFGS method, it cannot be used for the LM actor update.

5-2-2 Adaptive Simultaneous Perturbation Method

In Spall (2000) a method is devised that can estimate the Hessian by a single perturbation, the Adaptive Simultaneous Perturbation (ASP) approach. ASP can cope with a stochastic gradient. The Hessian is estimated by:

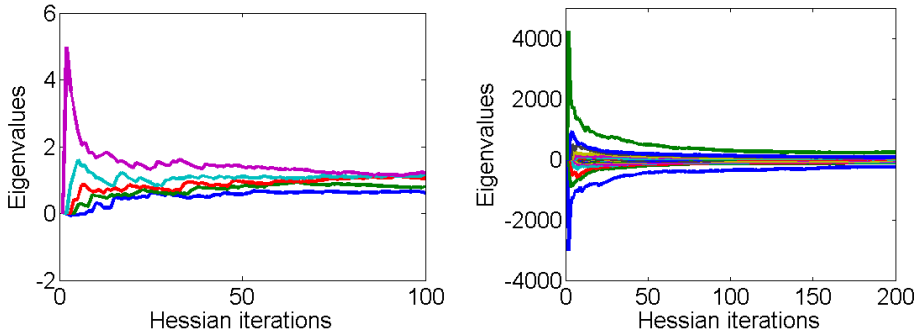
$$\tilde{H}_k = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{2}\left[\frac{\delta G_k^T}{2\chi_k\Gamma_k} + \frac{\delta G_k^T}{2\chi_k\Gamma_k}^T\right], \tag{5-21}$$

$$\delta G_k = \frac{\partial f(x_k + \chi_k\Gamma_k)}{\partial x} - \frac{\partial f(x_k - \chi_k\Gamma_k)}{\partial x}, \tag{5-22}$$

where χ_k is a positive scalar and $\Gamma_k \in \mathbb{R}^n$ a zero-mean random vector that satisfies certain regularity conditions. In this thesis a Bernoulli ± 1 variable is used. The used "vector-divide" operation in Equation (5-21) produces a matrix where the ij-th element is the ratio between the j-th element of the numerator row vector to the i-th element of the denominator column vector. To test the method it is first applied to a system with known Hessian:

$$y = \frac{1}{2}x^T I x \tag{5-23}$$

with $x \in \mathbb{R}^n$. In this example the dimension is set to $n = 5$. The gradient is corrupted with white noise with $\sigma = 0.1$ and the size of the perturbation is set to $\chi = 0.1$, so the noise and



(a) Identity matrix.

(b) $H(\vartheta^*)$ of the performance of the inverted pendulum.

Figure 5-1: Estimation of eigenvalues using ASP.

the perturbation are of the same magnitude. The estimation of the eigenvalues is shown in Figure 5-1a. ASP finds a reasonable estimate of the eigenvalues within approximately 100 gradient evaluations. To see how well ASP can be used in reinforcement learning (RL), it is tested on the inverted pendulum . The gradient is learned in a single episode using LSTD-Q for ϑ^*. The perturbation parameter is chosen to be $\chi = 0.1$. Smaller values lead to a slower Hessian estimation while bigger values have shown to give a poorer Hessian estimation. The results are shown in Figure 5-1b. ASP converges in about 200 iterations. The learned Hessian is not negative definite. Spall (2000) suggest to subtract a positive definite matrix, but due to the size of the eigenvalues this matrix needs to be rather big, destroying the information contained in the Hessian and resulting in small update steps.

5-3 Conclusion

To construct an LM actor update, a proper estimate of the Hessian $\frac{\partial^2 \rho}{\partial \vartheta^2}$ is needed. A number of different approaches have been taken to obtain an estimate of the Hessian. Unfortunately. no proper method could be found that estimates the Hessian quickly enough to beat regular policy gradient methods. The main difficulties are the variance in the gradient estimate and the sparse actor BF vector.
Even if a proper Hessian estimate could be found it does not have to be positive definite (see also the discussion in Kakade (2001a)). In Furmston and Barber (2012) a method is devised that estimates a Hessian that is guaranteed to be negative definite. The method has not been extended to the infinite time actor-critic framework yet. More research is needed to see if the approximate Hessian of Furmston and Barber (2012), or its largest eigenvalue, can be used to determine a proper actor learning rate in actor-critic RL.

Chapter 6

Exploiting Model and Reward Function Knowledge

Although reinforcement learning is a model-free technique, algorithms that learn a model of the environment have proven to increase learning speed tremendously (Sutton and Barto, 1998). Based on the potential to enhance reinforcement learning (RL) algorithms with the model learning techniques introduced in (Grondman et al., 2012b), a critic-only method has been devised that selects its actions by optimization of the right-hand side of the Bellman equation. This chapter is written as a paper about this method. Two weeks before the deadline of the project it was discovered that a very similar method already has been introduced in Doya (2000). The contribution of this thesis is to the work presented in Doya (2000) are:

- The implementation of a non-parametric process model approximator that is learned online.
- The evaluation of multiple critic functions.
- A comparison with other RL algorithms.
- Simulation and experimental tests on different systems.

The paper is written to highlight the additional gained insights.

6-1 Abstract

Reinforcement learning (RL) is a framework that enables a controller to find an optimal control policy for a task in an unknown environment. Although RL has been successfully used to solve a wide variety of control problems, learning is in general slow. The main causes are inefficient use of information collected during system interaction and not utilizing

available a priori knowledge about e.g. the system. We discuss applications of the Value-Gradient Based Policy (VGBP) algorithm that achieves faster learning by utilizing knowledge about the reward structure and the environment. The agent is provided information about its goal by giving it direct access to the reward function. Furthermore, the agent learns a process model. This enables the algorithm to select the optimal action by optimizing over the right-hand side of the Bellman equation. This paper utilizes function approximators that yield maximal performance with minimal dependence on initial parameters. We devise a learning rate free version of VGBP and demonstrate on the inverted pendulum swing-up task as well as a 2-DOF regulator task that the algorithm achieves fast learning.

6-2 Introduction

Reinforcement learning (RL) is a control methodology inspired by animal learning that enables an agent to become proficient in a task in an unknown environment by trial-and-error (Lewis and Vrabie, 2009). Feedback on the behavior is provided to the agent by a reward signal. The RL controller tries to optimize the cumulative reward signal, called the return. The expectation of the return is captured in the value function (Sutton and Barto, 1998). The control problem is solved as a sequential decision making problem. In each step the optimal control policy takes the action that yields the highest return. Since we consider RL in continuous environments, both the value function (the critic) and the policy (the actor) are estimated by function approximators.

One of the main disadvantages of RL is the slow learning caused by inefficient use of samples (Adam et al., 2012). Numerous methods have been devised to improve sample efficiency, like eligibility traces (Sutton and Barto, 1998), experience replay (ER)(Adam et al., 2012; Wawrzynski, 2009) and methods that learn a process model (Grondman et al., 2012b; Sutton, 1990). The model learning RL algorithms are related to approximate dynamic programming (DP) (Powell, 2007). In the Dyna (Sutton, 1990) algorithms the model is used to increase the learning speed of the value function. With the model additional system interaction is simulated. The simulated system interaction is then used to update the value function. The model learning actor-critic (MLAC) and reference model actor-critic (RMAC) algorithms presented in Grondman et al. (2012b) use a learned process model in efficient actor updates that lead to faster learning. In RMAC the model is used to train a reference model, which contains a mapping from state to desired next state. The action needed to go to the desired state is obtained with the local inverse of the learned process model. MLAC uses a model solely in the actor update. Like in Heuristic Dynamic Programming (HDP) (Si et al., 2004) the model is used to estimate the policy gradient. The actor is updated with a gradient ascent update rule.

The success of ER and model learning RL methods gives rise to the question if there is more information that can be provided to the agent to enable it to take better decisions. In the standard RL framework both the reward function and the system are considered to be in the environment (Sutton and Barto, 1998). The latter part is addressed by the model learning methods. In most cases the reward function is designed by an engineer. In this case the reward function can be easily provided to the agent. With the reward function and a learned process model the agent has a full understanding of the environment. This full view enables the agent to select optimal actions. An RL method that selects its actions with this full view is the Value-Gradient Based Policy (VGBP) algorithm (Doya, 2000). It selects the action by

optimizing the right-hand side of the Bellman equation. Because only the process model and the critic have to be learned and not an explicit actor, it is a critic-only method. VGBP has a number of favorable properties: by efficient use of the available information it achieves fast learning. It is a critic-only method that is able to generate a smooth continuous policy that is (very close to) the optimal signal. When the system dynamics are linear and the reward function is quadratic, the agent generates the same actions as a linear quadratic regulator (Doya, 2000). Furthermore prior knowledge about the system dynamics can be incorporated by initialization of the process model.

The contributions of this paper are threefold. First we apply different function approximation techniques to VGBP and devise a learning rate-free version of the algorithm. Second, we compare the learning behavior of VGBP with related RL algorithms to see the benefits of the incorporation of the reward function into the agent. Third, we show that VGBP not only achieves quick learning in simulation, but that the algorithm also works well on physical setups .

The used theoretical framework is introduced in Section 6-3. Related RL algorithms are described in Section 6-4. Function approximators used for the critic and the process model are presented in Section 6-5. In Section 6-6 the VGBP algorithm is introduced. The performance of VGBP is tested and compared with the on-policy critic-only method Sarsa and the model based MLAC algorithm on two systems in Section 6-7. A discussion about the learning behavior of VGBP is given in Section 6-8. Section 6-9 concludes the paper.

6-3 Markov Decision Process

We adopt the standard reinforcement learning formulation in which the agent environment interaction is modeled with a deterministic Markov decision process (MDP). An MDP is a four-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S} \in \mathbb{R}^D$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function.
At each time step k the agent receives the current state of the environment s_k. The agent uses a policy $\pi : \mathcal{S} \to \mathcal{A}$ to select actions. The action leads to a state transition $s_{k+1} = \mathcal{P}(s_k, a_k)$ and a reward $r_{k+1} = \mathcal{R}(s_k, a_k)$. The goal of the agent is to maximize the discounted sum of rewards, known as the return ρ. The discounted return of a policy is denoted by

$$\rho = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1}\right\}, \tag{6-1}$$

where $\gamma \in [0, 1)$ is the discount factor. An expectation of the return can be made for state s, or a state-action pair s, a. The expected return under policy π when starting in a state s is captured in the value function $V^\pi : \mathcal{S} \to \mathbb{R}$ denoted by

$$V^\pi(s) = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} \middle| s_0 = s\right\}. \tag{6-2}$$

The return received after taking action a in state s is stored in the Q-function $Q(s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, given by

$$Q^\pi(s, a) = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} \middle| s_0 = s, \quad a_0 = a\right\}. \tag{6-3}$$

Both the value and the Q-function satisfy the Bellman equation (Sutton and Barto, 1998):

$$V^{\pi}(s) = E\left\{\mathcal{R}(s,\pi(s)) + \gamma V^{\pi}(s')\right\}, \tag{6-4}$$

$$Q^{\pi}(s,a) = E\left\{\mathcal{R}(s,a) + \gamma Q^{\pi}(s,\pi(s'))\right\}, \tag{6-5}$$

where s' is the next state generated with the transition function $s' = \mathcal{P}(s,a)$. The relation between the value function and the Q-function is given by

$$Q^{\pi}(s,a) = \mathcal{R}(s,a) + \gamma V^{\pi}(s'). \tag{6-6}$$

The optimal policy π^* is the policy that has the highest return $V^*(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}$ and $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$. The optimal value functions satisfy the Bellman optimality equations

$$V^*(s) = \max_{a}\left(\mathcal{R}(s,a) + \gamma V^*(s)\right), \tag{6-7}$$

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \max_{a'} Q^*(s',a'). \tag{6-8}$$

The optimal policy π^* can be derived from the optimal Q-function $Q^*(s,a)$ by taking in each state the action with the highest return.

$$\pi^*(s) \in \arg\max_{\mathcal{A}} Q(s,a). \tag{6-9}$$

This policy is *greedy* in the Q-function.

6-4 Solving the Markov Decision Process

In this section two algorithms that solve an MDP are introduced that are comparable to VGBP. The methods learn the value function of the followed policy, they are *on-policy* algorithms. The most well known on-policy critic-only method is Sarsa. This method is used to compare VGBP to general on-policy critic-only methods. Thereafter MLAC is introduced. Both MLAC and VGBP use the process model to obtain the gradient of the return with respect to the action $\frac{\partial \rho(s)}{\partial a}$. Because this similarity, a comparison between MLAC and VGBP provides more insight in the utilization of a process model and a reward function in continuous reinforcement learning.

6-4-1 Sarsa

Sarsa selects actions based on the Q-function. In the adopted approximate RL framework the state-action value function is the parameterized function $Q^{\pi}_{\theta}(s,a)$ with parameter vector $\theta \in \mathbb{R}^p$. The Q-function is learned by minimizing the difference between the left and right hand sized of the Bellman Equation (6-5). This mismatch is captured in the temporal difference error (TD) δ_{k+1}:

$$\delta_{k+1} = r_{k+1} + \gamma Q_{\theta_k}(s_{k+1}, a_{k+1}) - Q_{\theta_k}(s_k, a_k). \tag{6-10}$$

The name Sarsa is an acronym for the used tuple: (s, a, r, s', a'). The Q-function is learned by minimizing the squared TD-error, for instance by gradient descent:

$$\theta_{k+1} = \theta_k + \alpha_{c,k}\delta_{k+1}\frac{\partial Q_{\vartheta}(s_k,a_k)}{\partial \theta}, \tag{6-11}$$

where $\alpha_{c,k} > 0$ is the critic learning rate. To guarantee convergence the learning rate have to satisfy the Robbins-Monro conditions (Robbins and Monro, 1951):[1]

$$\alpha_k > 0 \quad \forall k \qquad \sum_k \alpha_k = \infty \qquad \sum_k \alpha_k^2 < \infty. \tag{6-12}$$

Furthermore all state-action pairs need to be visited infinitely often. This requirement has to be incorporated in the policy that is used to find the value function. This can be ensured by sometimes taking exploration actions instead of greedy actions. A simple exploration method is *ϵ-greedy action selection*: with probability $1-\epsilon$ (where $\epsilon \in (0,1)$) the greedy action is taken and with probability ϵ a random action is chosen from $\mathcal{A}$. Approximate Sarsa converges with probability 1 if the exploration term ϵ decays to zero and the policy satisfies certain regularity conditions (Melo et al., 2008).
The update Equation (6-11) assigns the received reward r_{k+1} only to the latest transition. But the current reward is the result of the entire trajectory. Learning can be sped by using *eligibility traces* $e \in \mathbb{R}^n$ that allow the current reward r_{k+1} to update the Q-function of recently visited states. The update equation with eligibility traces is

$$e_k = \lambda e_{k-1} + \frac{\partial Q_\vartheta(s_k, a_k)}{\partial \theta}, \tag{6-13}$$

$$\theta_{k+1} = \theta_k + \alpha_k \delta_{k+1} e_k, \tag{6-14}$$

where $\lambda \in [0,1]$ is the trace decay parameter and $e_0 = 0$.

6-4-2 Model Learning Actor-Critic

MLAC consists of three parts; it contains a separate function that describes the policy π (the actor), an element that evaluates the policy (the critic) in the form of a value function and a process model (Grondman et al., 2011b). The critic and the process model steer the actor towards an optimal policy. The critic is updated according to

$$\delta_{k+1} = r_{k+1} + \gamma V_\theta(s_{k+1}) - V_\theta(s_k), \tag{6-15}$$

$$e_k = \lambda e_{k-1} + \frac{\partial V_\theta(s_k)}{\partial \theta}, \tag{6-16}$$

$$\theta_{k+1} = \theta_k + \alpha \delta_{k+1} e_k, \tag{6-17}$$

with $e_0 = 0$. The policy is the parameterized function π_ϑ with $\vartheta \in \mathbb{R}^p$. The policy is updated with the following gradient ascent update rule

$$\vartheta_{k+1} = \vartheta_k + \alpha_{a,k} \frac{\partial \rho(s_k)}{\partial \vartheta}, \tag{6-18}$$

where $\alpha_{a,k}$ is the actor step size that has to satisfy the Robbins Monro conditions and $\frac{\partial \rho}{\partial \vartheta}$ is the policy gradient. With the process model $\hat{\mathcal{P}}$, the policy gradient is approximated by

$$\frac{\partial \rho(s)}{\partial \vartheta} \approx \frac{\partial V_\theta(s)}{\partial s} \frac{\partial \hat{\mathcal{P}}}{\partial a} \frac{\partial \pi_\vartheta(s)}{\partial \vartheta}. \tag{6-19}$$

[1] The subscribt c is dropped because these are general step size conditions that also apply to the actor learning rate.

6-5 Function Approximation

In this paper two types of function approximators are used: local linear regression (LLR) and approximators that are linear in the parameters (LIP).

6-5-1 Local Linear Regression

LLR is a non-parametric method that approximates a nonlinear function $y = f(x)$, with input vector $x \in \mathbb{R}^n$ and output vector $y \in \mathbb{R}^m$ by fitting a locally affine model to stored input-output data. The output for a given input x is then approximated by

$$\hat{y} = \beta \begin{bmatrix} x \\ 1 \end{bmatrix}, \tag{6-20}$$

where β is the model parameter matrix. During learning, observations of input-output data, called samples, are collected in the memory. The samples are column vectors $\varsigma_i = [x_i^T | y_i^T]^T$, with $i = 1 \dots N$. The samples are stored in the columns of the memory matrix M, of size $(m + n) \times N$. Outdated samples have to be replaced with new samples. The memory management is beyond the scope of this paper. LLR is referred to as a lazy classifier, because it simply stores all the input data and only performs a computation when a query is made (Wettschereck et al., 1997). To obtain the process model β, first the K nearest samples in the memory are selected, according to a weighted distance metric (Grondman et al., 2012b). The input is scaled with a scaling matrix W. This matrix has a large influence on the accuracy of the prediction (Wettschereck et al., 1997). The input and output data of the nearest neighbors are collected in the matrices

$$Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_K \end{bmatrix} \qquad X = \begin{bmatrix} x_1 & x_2 & \cdots & x_K \\ 1 & 1 & \cdots & 1 \end{bmatrix}. \tag{6-21}$$

The model parameter β is found by minimizing the squared error $\varepsilon = |Y - \beta X|^2$. This can for instance be done with the right pseudo-inverse:

$$\beta = YX^T(XX^T)^{-1}. \tag{6-22}$$

Advantages of LLR are that no global structure needs to be defined, broad generalization and faster initial learning than incremental methods.

6-5-2 Linearly Parameterized Approximators

In this paper linearly parameterized approximators are used to approximate the value function. A linear parametrization is used because the theoretical properties are easy to analyze (Buşoniu et al., 2010). An approximation of a value function that is linear in the parameters is denoted by

$$V_\theta^\pi(s) = \theta^T \phi(s) \qquad \text{or} \qquad Q_\theta^\pi(s,a) = \theta^T \phi(s,a), \tag{6-23}$$

where $\theta \in \mathbb{R}^p$ is the parameter vector and $\phi(s) : \mathcal{S} \to \mathbb{R}^p$ and $\phi(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^p$ are vetors of basis functions (BFs). They are a useful tool to incorporate a priori knowledge of the

designer about the problem (Bertsekas and Tsitsiklis, 1995). In this paper Gaussian radial basis functions (RBFs) are used. Normalized Gaussian RBFs are denoted by

$$\phi_i(s) = \frac{\bar{\phi}_i(s)}{\sum_{i'=1}^{p} \bar{\phi}_{i'}(s)}, \tag{6-24}$$

$$\bar{\phi}_i(s) = \exp\left(-\frac{1}{2}(s - c_i)^T B_i^{-1}(s - c_i)\right), \tag{6-25}$$

where $\bar{\phi}_i(s)$ is the non normalized BF. The center of the RBF is given by the vector $c_i = [c_{i,1}, c_{i,2}, \ldots, c_{i,D}]^T \in \mathbb{R}^D$. The width is determined by the symmetric positive definite matrix $B \in \mathbb{R}^{D \times D}$.

6-6 Algorithm

Critic-only algorithms select the greedy action a by finding the maximum of the left hand side of Equation (6-6). The right hand side of Equation (6-6) can be used to select the action if both the reward function and the transition model are known. In most cases the reward function is designed by an engineer and can therefore be provided to the agent. By learning a process model $\hat{\mathcal{P}}$ and with the accessible reward function, the optimal action is found by solving

$$a = \arg\max_a \left(\mathcal{R}(s, a) + \gamma V_\theta\left(\hat{\mathcal{P}}(s, a)\right)\right). \tag{6-26}$$

Because both $\mathcal{R}(s, a)$ and $V_\theta(s)$ are in general highly nonlinear, Equation (6-26) can be hard to solve. If V_θ is smooth enough to be approximated by first order Taylor series and the process model is approximately linear in the control action, the Q-function can be approximated by

$$Q(s, a) \approx \mathcal{R}(s, a) + \gamma V_\theta(s) + \gamma \frac{\partial V_\theta(s)}{\partial s} \frac{\partial s}{\partial a} a. \tag{6-27}$$

The nonlinearity in the reward function remains. If the reward function is (piecewise) convex, a unique (local) solution can be easily found. This can for instance be done by setting the derivative of Equation (6-27) to zero:

$$-\frac{\partial \mathcal{R}(s, a)}{\partial a} = \gamma \frac{\partial V_\theta(s)}{\partial s} \frac{\partial \hat{\mathcal{P}}}{\partial a}. \tag{6-28}$$

The computed action, may lay outside the action space $\mathcal{A}$ and needs therefore to be clipped to stay inside $\mathcal{A}$. To guarantee exploration a zero-mean noise term ϵ is added to the clipped action and the action plus exploration needs to be clipped again.

6-6-1 Process Model

The process model $s' = \hat{\mathcal{P}}(s, a)$ is learned with LLR. This approximator is picked because it has shown to learn a good process model with few observations. Samples $\zeta_i =$

$\begin{bmatrix} s_i^T & a_i^T & | & s'^T_i \end{bmatrix}$ of the process are stored in the memory M^P. The transition caused by action a in state s is predicted with

$$s' = \begin{bmatrix} \beta_s^P & \beta_a^P & \beta_b^P \end{bmatrix} \begin{bmatrix} s \\ a \\ 1 \end{bmatrix}. \tag{6-29}$$

The superscript P denotes that it is the process model and the subscripts state the input variable with which it is multiplied The derivate of the process model needed in Equation (6-28) is then simply $\frac{\partial \hat{\mathcal{P}}}{\partial a} = \beta_a^P$.

6-6-2 Critic Parametrization

VGBP can use any critic function that is differentiable with respect to the state. In this paper two critic approximators are used: the LLR critic of Grondman et al. (2012b) and a LIP approximator using RBFs. The LLR critic collects the samples $\zeta_i = [s_i^T | V_i]^T$ with $i = 1, \ldots, N^C$ in the critic memory M^C. The value of a state is computed with the local affine model

$$V(s) = \begin{bmatrix} \beta_s^C & \beta_b^C \end{bmatrix} \cdot \begin{bmatrix} s \\ 1 \end{bmatrix}. \tag{6-30}$$

For the local affine value function approximation the right hand side of Equation (6-28) reduces to $\gamma \beta_s^C \beta_a^P$. The Equations of the RBF critic are given in Section 6-5-2. The derivative of the value function with respect to the state is given by

$$\frac{\partial V_\theta(s)}{\partial s} = \theta^T \frac{\partial \phi(s)}{\partial (s)}. \tag{6-31}$$

For a diagonal scaling matrix $B = \text{diag}[b_1, \ldots, b_D]$ as used in this paper we have

$$\frac{\partial \phi_i(s)}{\partial s} = -\phi_i(s) \left[B^{-1}(s - c_i) + \frac{\sum_j \frac{\partial \bar{\phi}_j(s)}{\partial s}}{\sum_j \bar{\phi}_j(s)} \right]. \tag{6-32}$$

Because the RBF approximator is LIP, the parameter vector θ can also be obtained by least squares. The Least Squares Temporal Difference (LSTD) algorithm (Boyan, 2002) is denoted by:

$$e_{k+1} = \lambda e_k + \phi(s_k), \tag{6-33}$$
$$A_{k+1} = A_k + e_k(\phi(s_k) - \gamma \phi(s_{k+1}))^T, \tag{6-34}$$
$$b_{k+1} = b_k + e_k r_{k+1}, \tag{6-35}$$
$$\theta_{k+1} = A_{k+1}^{-1} b_{k+1}, \tag{6-36}$$

where $e_0 = \phi(s_0)$, $b_0 = 0$ and A_0 is chosen as a small invertible matrix. The advantages of the LSTD critic are that there is no learning rate (schedule) to be tuned and the high learning speed. A disadvantage is the increased computational cost.
The full algorithm for VGBP using LLR and a general critic is given in Algorithm 6.1.

Algorithm 6.1 Value-Gradient Based Policy with LLR process model

Input: : γ,λ,σ

initialize $k = 0$, s_0, θ, M^P, $e_k = 0$
draw action $a \sim \mathcal{N}(0, \sigma)$ and clip it
for every step $k = 1, 2, \ldots$ **do**
  measure s_{k+1} and compute reward r_{k+1}
  Critic
  $\delta_{k+1} \leftarrow r_{k+1} + \gamma V^\pi(s') - V^\pi(s)$
  $e = \lambda e + \phi(s_k)$
  update critic
  Process model
  Insert $\begin{bmatrix} s_k^T & a_k^T & | & x_{k+1}^T \end{bmatrix}^T$ in M^P
  Action selection
  Obtain β^P from M^P for s_{k+1}
  find a_c by solving $-\frac{\partial \mathcal{R}(s_{k+1}, a, s')}{\partial a} = \gamma \frac{\partial V_\theta(s_{k+1})}{\partial s} \beta_a^P$
  draw $\epsilon \sim \mathcal{N}(0, \sigma)$
  $a = \text{clip}(\text{clip}(a_c) + \epsilon)$
end for

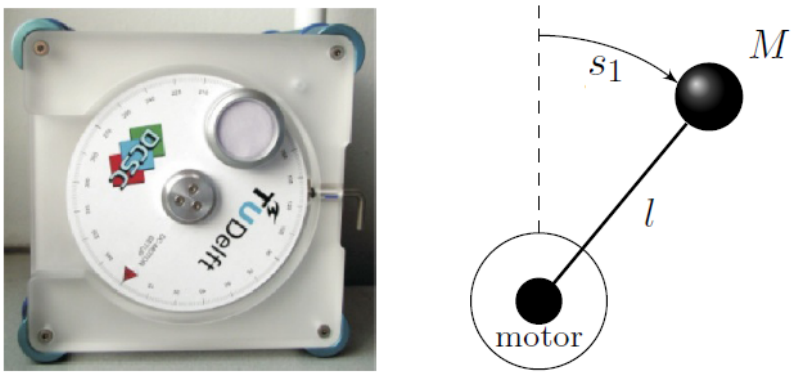


Figure 6-1: The inverted pendulum.

6-7 Simulation and Experimental Results

The working principles of VGBP are verified on the inverted pendulum swing-up task in simulation as well as on a setup in Section 6-7-1. This task is a standard benchmark in RL (Buşoniu et al., 2010). The learning behavior of VGBP in a multidimensional action space is tested on a two link robotic manipulator in Section 6-7-2.

6-7-1 Inverted Pendulum

A picture of the setup is given in Figure 6-1. The goal is to swing up a pendulum from a down-pointing position to the upright position as quickly as possible and keep the pendulum upright by applying a voltage to the DC-motor. The action space is limited to $a \in [-3, 3]$ V,

Table 6-1: Parameters of the inverted pendulum.

Model parameter	**Symbol**	**Value**	**Units**
pendulum inertia	J	$1.91 \cdot 10^{-4}$	kgm ^{2}
pendulum mass	M	$5.50 \cdot 10^{-2}$	kg
gravity	g	9.81	m/s^2
pendulum length	l	$4.20 \cdot 10^{-2}$	m
damping	b	$3.0 \cdot 10^{-6}$	Nms
torque constant	K	$5.36 \cdot 10^{-2}$	Nm/A
rotor resistance	R	9.50	Ω
RL parameter	**Symbol**	**Value**	**Units**
sampling time	T_s	0.03	s
discount factor	γ	0.97	
exploration noise	σ^2	1	V
maximum action	$a_{\max}$	3	V
minumum action	$a_{\min}$	-3	V

which makes it impossible to steer the pendulum directly from the start state to its upright position. Instead the agent needs to learn to swing the pendulum to the top position.
Because MLAC and VGBP use the same value gradient, they are compared first. Thereafter VGBP is compared with Sarsa. Finally, MLAC using an LSTD critic is discussed.

Setup

The dynamics of the pendulum are given by

$$\begin{bmatrix} \dot{s}_1 \\ \dot{s}_2 \end{bmatrix} = \begin{bmatrix} s_2 \\ \frac{1}{J}\left(Mgl\sin(s_1) - \left(b + \frac{K^2}{R}s_2\right)\right) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{K}{JR} \end{bmatrix} a. \tag{6-37}$$

where s_1 is the angle of the pendulum and s_2 the angular velocity. The model parameters are stated in Table 6-1. The angular velocity signal provided to the agent is saturated to stay in $[-25, 25]$ rad/s. The reward is calculated with a quadratic reward function

$$r_k(s_k, a_{k-1}) = -s_k^T Q s_k - a_{k-1}^T P a_{k-1}, \tag{6-38}$$

with

$$Q = \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix} \qquad P = 1.$$

The learning procedure is split into trials of 3 seconds after which the pendulum is reset to its downward position ($[\pi \quad 0]^T$). A learning experiment consist of 200 trials. A sampling time of $0.03s$ is used. All simulations and experiments are repeated 20 times to acquire the average learning behavior as well as the confidence region of the mean.

Table 6-2: Agent parameters simulated inverted pendulum for MLAC and VGBP with LLR approximators.

Parameter		Actor	Critic	Model
learning rate	α	0.04	0.1	1
trace decay factor	λ	0	0.65	0
memory size	N	2000	2000	2500
nearest neighbors	K	9	20	30
input scaling	$\text{diag}(W)$	[1 0.1]	[1 0.1]	[1 0.1 1]

Model Learning Methods

Because MLAC and VGBP differ only in the action selection, the result of our new algorithm can be made clearly visible by giving both algorithms the same process model and critic approximator. LLR is used in both the process model and the critic, because this approximator has shown to yield the best performance (Grondman et al., 2012a). The parameters are stated in Table 6-2. Given the reward function (6-38), the desired action for VGBP is given by

$$a = \frac{1}{2}\gamma P^{-1}\frac{\partial V_\theta(s)}{\partial s}\beta_a^P. \tag{6-39}$$

An exploratory action is applied for both MLAC and VGBP once every three time steps, where $\varepsilon \sim \mathcal{N}(0,1)$. On the one hand this exploration strategy generates large exploratory actions, which force the agent out of the known part of the state space. On the other hand the policy is not dominated by the exploration noise and the agent gets time to correct faulty exploratory actions.

The achieved learning behavior including confidence bounds of MLAC is shown in Figure 6-2a. Learning converges in approximately 9 minutes of interaction time. The confidence region for the mean is very narrow, indicating that a steady swing-up is achieved each time. The learned policy is given in Figure 6-2b. The results of VGBP are given in Figure 6-2c. After 25 trials the learning speed increases, because VGBP has learned to get the pendulum to its top position, but not yet how to stabilize the pendulum there. The algorithm converges after 3 minutes of learning. The final policy is slightly worse than that of MLAC, as VGBP needs two or three swings to get the pendulum upright. MLAC only needs two. A learned policy is shown in Figure 6-2d.

Sarsa

To see how the learning behavior of VGBP relates to critic-only methods, Sarsa is applied to the inverted pendulum. The critic is LIP, with a set of 400 RBFs . The centers are positioned on a 20×20 equidistant grid over the state space. All RBFs have the same scaling matrix B. The scaling matrix is chosen such that the value of the non-normalized BF is 0.5 at the center of each neighboring BF. The action space of Sarsa is $\mathcal{A}_{\text{Sarsa}} = \{-3, 0, 3\}$. Each action has the same set of BFs, denoted by $\phi_b(s) \in \mathbb{R}^{400}$. This results in a parameter vector $\theta \in \mathbb{R}^p$ with $p = 3 \cdot 400 = 1200$. For each state-action pair only the part of the BF that is concerned with the selected action is nonzero. For instance $Q(s,0) = \begin{bmatrix}[0^T & \phi_b(s)^T & 0^T\end{bmatrix}\theta$, were $0 \in \mathbb{R}^{400}$ is

the zero vector.
The actions are selected with ϵ-greedy action selection, where ϵ is determined by

$$\epsilon = \max\left(0.1, 0.96^{\text{trial}-1}\right). \tag{6-40}$$

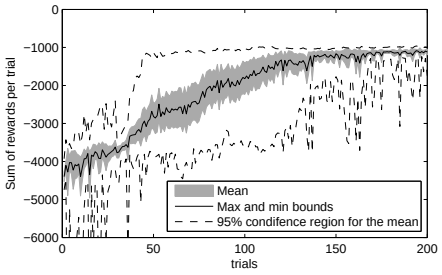
The decay factor 0.96 and the minimum exploration probability were found by manual tuning. The critic learning rate is set to $\alpha_c = 0.5$ and the trace decay factor to $\lambda = 0.85$. The results are shown in Figure 6-2g. Sarsa needs more system interaction than the model learning algorithms. The final performance is slightly better than to that of VGBP-LLR.

LSTD-Critic

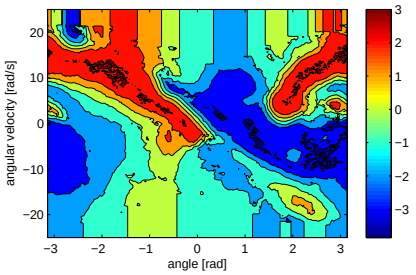
To remove the dependency of VGBP on the critic learning rate, LSTD is applied. The same set of 400 equidistant BFs is used as for Sarsa. The initial A-matrix is set to $A = 0.5I$ and the b-vector to $b_0 = 0$. This gives the value function a bias towards 0. It is an optimistic initialization, since all rewards are negative. This stimulates exploration in the early phase of learning and prevents overfitting of the LSTD. The eligibility factor is set to $\lambda = 0.3$. With this relatively low value for the trace decay factor λ the optimistic initialization is contained longer than for bigger values of λ. Tests indicated that this is the optimal mix between stimulating exploration and propagating the reward to recently visited states. The learning is further sped up by adding exploration noise to the action at every time step. The critic parameter is updated at the end of every trial. This gives more steady behavior than when the policy is updated after each time step. The achieved average learning is depicted in Figure 6-2e. A swing-up is learned in less than a minute of system interaction. A typical trajectory generated with a learned policy is shown in Figure 6-3. The agent pulls the pendulum with maximum voltage to one side. As the maximum angle reached the agent reverses the control action to create the swing to the top. Because the control action is cheap compared to the penalty on the angle, the agent adds more energy to the system than strictly needed for a swing-up. This is done to leave the expensive downwards position as quickly as possible. The additional energy is dissipated by the damping in the final stage of the swing-up (after 0.7 s). In Figure 6-2h the average learning behavior of all the above methods is compared. As can be seen VGBP does not only learn faster with an LSTD-critic, but also finds a better, more steady policy. The quality of the learned policies of VGBP-LSTD and Sarsa is almost similar. The final policy of MLAC is the best.

Results on the Physical Setup

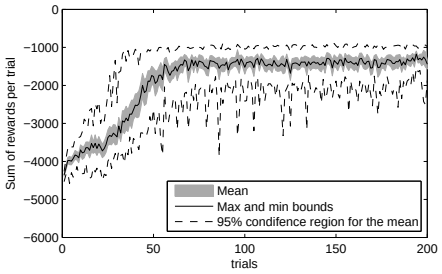
Due to time constraints not algorithms could be tested on the setup. Since in simulation the VGBP with the LSTD-critic performed the best, it is applied to the setup. Exactly the same agent is used. Because the setup measures only the angle, the model (6-37) is used as an observer. The action is determined with the predicted state of the model. This makes the time delay between state measurement and action execution minimal. This is needed to learn a proper process model.
The learning behavior of VGBP-LSTD on the setup is shown in Figure 6-4. Also the mean of the simulated learning curves is shown. Learning converges in 1.5 minutes. The results for the setup and the simulation are very similar. There are two main differences. On the



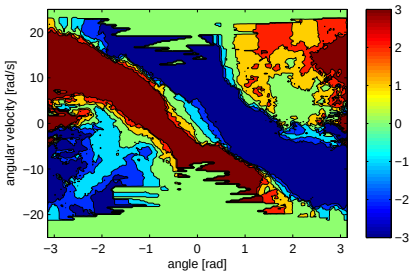
(a) Results of MLAC.



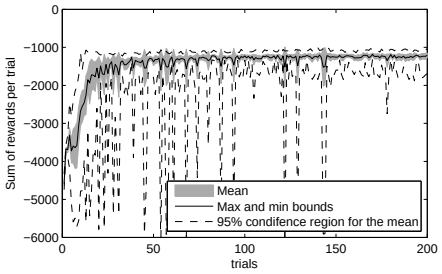
(b) Typical learned policy $\pi(s)$ by MLAC.



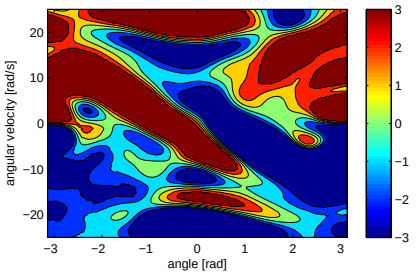
(c) Results of VGBP-LLR.



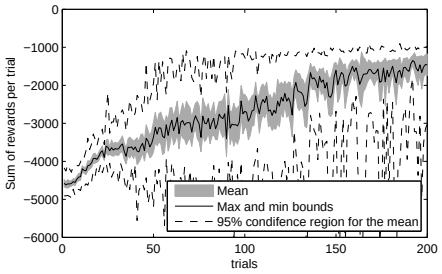
(d) Typical learned policy $\pi(s)$ by VGBP-LLR.



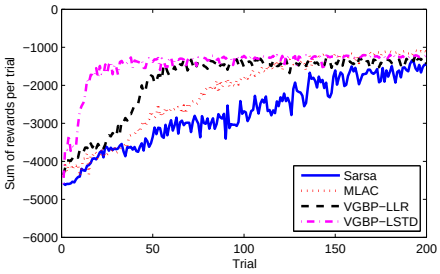
(e) Results of VGBP-LSTD.



(f) Typical learned policy $\pi(s)$ by VGBP-LSTD.



(g) Results of SARSA.



(h) Average learning behavior of SARSA, MLAC VGBP-LLR and VGBP-LSTD.

Figure 6-2: Results of 20 simulations of MLAC, VGBP-LLR, VGBP-LSTD and SARSA on the inverted pendulum swing-up task and typical learned policies.

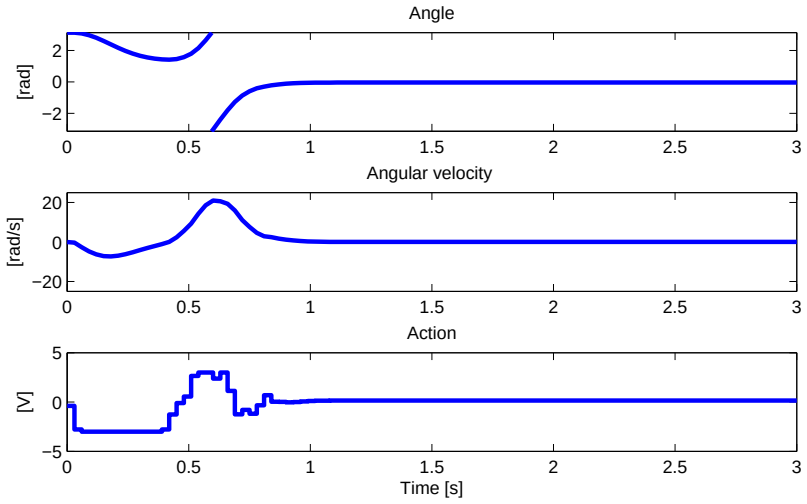


Figure 6-3: Trajectory of the swing-up of VGBP-LSTD in simulation.

setup VGBP needs only two swings to get the pendulum upright. Although the number of swings is smaller on the physical setup, the return does not increase, because there are some unmodeled dynamics in the setup. It takes slightly longer to complete a swing-up than in simulation. The other difference is that the spikes in the min-bound disappear on the setup after 30 trials, and not in the simulation results (Figure 6-2e).This is probably caused by the unmodeled inertia in the gear box of the motor. This makes the movements a fraction smoother on the setup than in simulation. The smoother behavior leads to less spikes in the return.

6-7-2 Robotic Manipulator

In this section VGBP learns to control a two-link manipulator to its center position. Both joints are actuated.

Setup

A picture and a schematic overview of the system are given in Figure 6-5. The model parameters are stated in Table 6-3. The system operates in a horizontal plane. It has four states, two angles and two angular velocities: $s = \begin{bmatrix}\psi_1 & \dot{\psi}_1 & \psi_2 & \dot{\psi}_2\end{bmatrix}^T$. The action a consists of the torques on the links $a = [\tau_1 \quad \tau_2]^T$. The absolute maximum action signal for both links is set to 0.1. Due to mechanical constraints the operating range is set for both links to be $[-1.2, 1.2]$ rad. Difficulties are the very high friction in the links an the coupling between the states. For this task the quadratic reward function (6-38) is used with

$$Q = \text{diag}\begin{bmatrix}200 & 1 & 50 & 1\end{bmatrix} \qquad P = \text{diag}\begin{bmatrix}2 & 2\end{bmatrix}$$

In case the manipulator is steered outside the operating range a reward of -4×10^4 is given and the trial is terminated. Trials last (at most) 10 seconds. The sampling time is set to

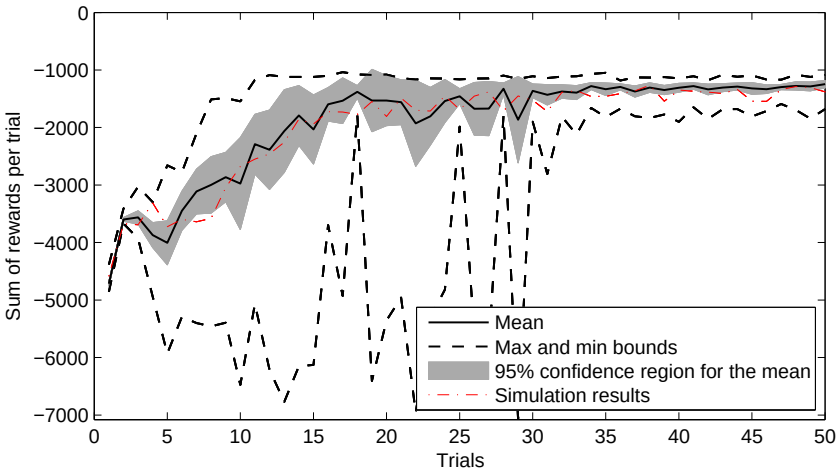


Figure 6-4: Performance of VGBP of 20 experiments on the pendulum setup.

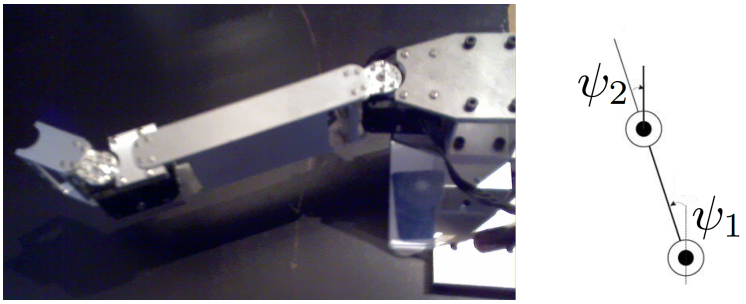


Figure 6-5: 2-DOF robotic manipulator.

Table 6-3: Parameters of the robot arm.

Model parameter	**Symbol**	**link 1**	**link 2**	**Units**
length	l	0.1	0.1	m
mass	m	0.125	0.05	kg
center off mass	m_c	0.04	0.06	m
inertia	I	0.074	1.2×10^4	kgm^2
RL parameter	**Symbol**	**Value**		**Units**
sampling time	T_s	0.01		s
discount factor	γ	0.97		
exploration noise	σ	0.05		
action space	$\mathcal{A}$	[-0.1,0.1]	[-0.1,0.1]	Nm

Table 6-4: Agent parameters of VGBP on the robotic manipulator.

Process Model		
trace decay factor	λ	0
memory size	N^P	5000
nearest neighbors	KP	80
input scaling	$\text{diag}(W)$	$\begin{bmatrix}1 & 0.05 & 1 & 0.05 & 5 & 5\end{bmatrix}$
Critic		
trace decay factor	λ	0.3
basis functions	N_{BF}	$\begin{bmatrix}5 & 3 & 5 & 3\end{bmatrix}$
angles	ψ_1; ψ_2	$[-1.2, 1.2]$; $[-1.2, 1.2]$ rad
angular velocities	$\dot{\psi}_1$; $\dot{\psi}_2$	$[-0.8, 0.8]$; $[-0.8, 0.8]$ rad/s
initial A- matrix	A_0	$0.5I$

$0.05s$. Learning is split into blocks of four trials, where the initial states of one block are

$$S_0 = \left\{ \begin{bmatrix}1\\0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\-1\\0\end{bmatrix}, \begin{bmatrix}-1\\0\\1\\0\end{bmatrix}, \begin{bmatrix}-1\\0\\-1\\0\end{bmatrix} \right\}$$

These states are the furthest from the goal state. Performance is measured by taking the average return per block. In this manner an estimate is made for the lower bound of the return when starting from a random initial state.

6-7-3 Agent

The value function approximation is LIP with RBFs. The centers are placed on an equidistant grid over the state space. This grid is defined for both links over $[-1.2, 1.2]$ rad and for the angular velocities over $[-0.8, 0.8]$ rad/s. The number of BF's along each dimension are $\begin{bmatrix}5 & 3 & 5 & 3\end{bmatrix}$, giving a total of 225 BFs. The B matrix is chosen in the same way as in Section 6-7-1. The critic is updated with LSTD(λ). The trace decay factor is set to $\lambda = 0.3$. The parameter vector θ is updated after the completion of a block. This proved to give the highest learning speed, since information of a bigger part of the state space is taken into account. Episodic LSTD updates have a higher tendency to overfit the critic on the last trajectory. The complete set of agent parameters is stated in Table 6-4.

6-7-4 Simulation Results

The results presented in this section are obtained using a simulation of the system with identified parameters. Figure 6-6 show the average learning behavior obtained from 20 learning experiments. A typical trajectory is shown in Figure 6-7. The algorithm is able to steer the arm close to its center position. Due to the very high friction and the long sampling time the assumption that the value function is linear with respect to the action is violated near the center position. This causes the agent to overshoot the middle position. The low minimum

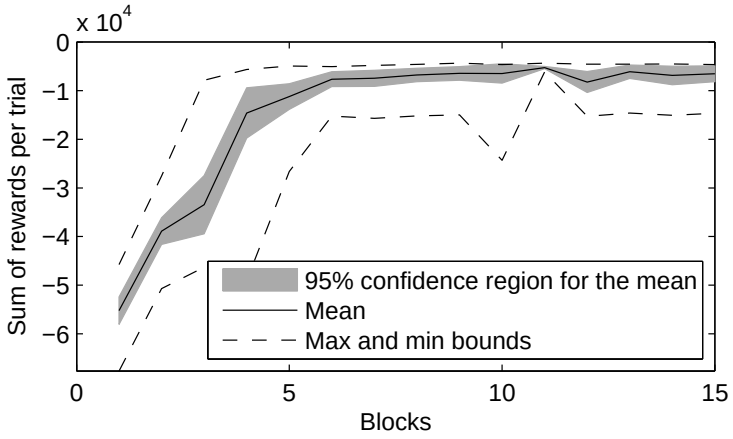


Figure 6-6: Results of VGBP on the 2-DOF manipulator in 20 simulations.

bound is the result of VGBP occasionally steering the arm out of the feasible region. This is caused by the optimistic initialization. Through exploratory actions the agent reaches unexplored parts of the state space. Because unseen states have a higher value than the known states, the agent leaves it's known area and heads towards unexplored (terminal) states. This can be prevented by giving the known terminal states a very low initial value. The experiment shows that VGBP can be easily applied to multidimensional learning problems, reaching a proper policy in a short amount of time.

6-8 Discussion

VGBP learns the value function, but uses its derivative in the action selection. For convex reward functions the optimal action is found by solving a convex optimization problem. In tasks where extreme actions are required for large parts of the state space, like the robotic manipulator, this proves to be very efficient, because the maximum action is taken when the gradient is steep and points loosely in the proper direction. With an optimistic initialization this results in rapid exploration of the state space, leading to quick learning. A disadvantage is that the variance in the value function estimates is amplified in the gradient and therefore in the action selection. This resulted in a less stable final policy on simulations of the inverted pendulum swing-up; the agent needed two or three swings to get the pendulum upright. Other algorithms needed the same amount of swings each time. Furthermore slower update schemes for the critic (per episode or per block) resulted in better performance. Too high update rates can result in overfitting, leading to suboptimal policies.
Other algorithms that use the gradient of the value function are MLAC and HDP methods. These methods suffer less from the variance in the value function because it is filtered by the actor update rule.

6-9 Conclusions and Future Work

In this paper we introduced a learning rate free version of VGBP. This method selects its actions via a direct optimization over the right-hand side of the Bellman equation. In con-

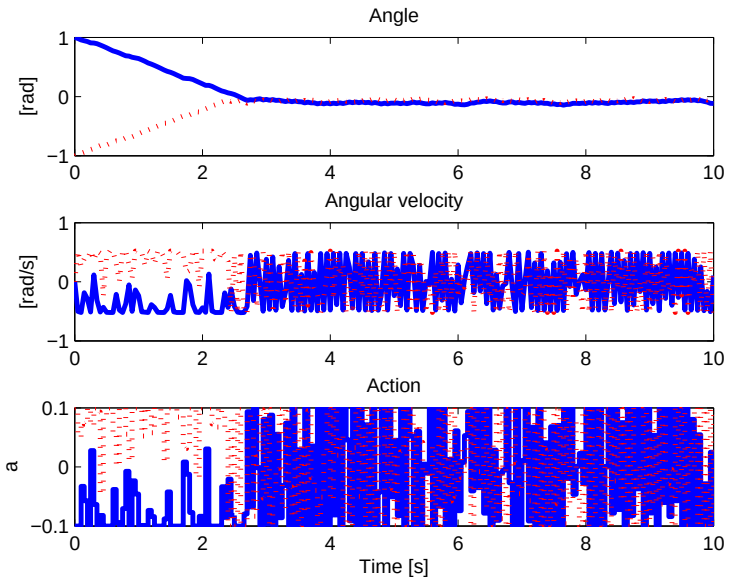


Figure 6-7: Typical learned trajectory on the manipulator. The solid blue line is ψ_1 and the red dash-dotted is ψ_2.

trast to regular RL methods the reward function is accessible to the agent. With a learned process model the agent can predict the result of its actions. We have shown that the access to the reward function enables VGBP to learn faster than another RL algorithm that uses a model to obtain the same value gradient. The quality of the policy is dependent on critic and the model approximation. We have shown that the framework works for multiple critics, both parametric and nonparametric. VGBP-LSTD showed the the highest learning speed. It achieves fast learning both in simulation and on physical systems. In addition no learning rates need to be tuned. In this paper LLR is used as function approximator for the process model. It is interesting to investigate if the model can not only be used in the action selection, but also for the critic training like in the Dyna algorithms. The two learning tasks in this paper have a quadratic reward function. Further research is needed to know if the good learning properties are contained for problems with more complex reward structures. As addressed in Section 6-7-4 the algorithm has a high tendency to explore unseen states, including bad terminal states. It is worth investigating how the value function could be optimally initialized with the reward function. This would allow to safely implement VGBP on learning tasks on mechanical systems with physical limitations.

Chapter 7

Additional Results of Value-Gradient Based Policy

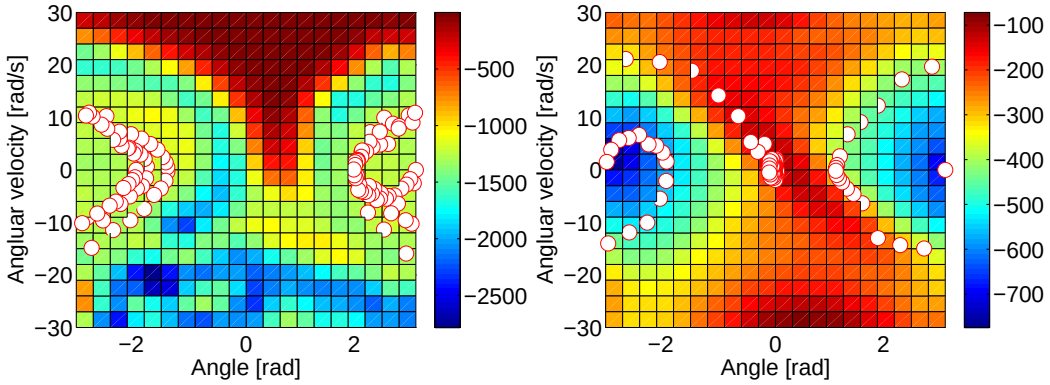
In this chapter additional results of experiments with Value-Gradient Based Policy (VGBP) are given. This chapter is split into two parts. First, experiments on the inverted pendulum are discussed. Then, results of VGBP on the 2-DOF robotic manipulator, from here on referred to as the "Goalkeeper", are treated.

7-1 The Inverted Pendulum

The most important critic parameters that have to be determined for a learning experiment with VGBP-LSTD(λ) are the initial critic and the trace decay factor λ. The influence of these parameters is discussed in this section.

7-1-1 Initialization of the LSTD-Critic

The initialization of the LSTD-critic consists of two parts: the matrix A_0 and the vector b_0. The initial critic parameter vector θ_0 is set to $\theta_0 = A_0^{-1}b_0$. For the LSTD-update the A-matrix has to be invertible. Often the matrix is chosen as $A_0 = qI$, where q is a very small constant. This constant has to be chosen big enough to keep A numerically invertible. The constant introduces a bias in θ towards $\theta = A_0^{-1}b_0$.
The vector b_0 is in most cases chosen as $b_0 = 0$. This initialization is also adopted in this thesis. This gives the initial parameter $\theta_0 = 0$. Because all the rewards are negative, this is an optimistic initialization. The agent prefers to visit unvisited states over known states, because the unexplored states have a higher value. This improves exploration and thereby the learning speed. With $b_0 = 0$ the parameter q determines the bias for θ towards 0. The bigger q, the longer the entries of θ will be close to zero. The choice of q has similarities to the choice of the learning rate.
The influence of the q is clearly seen in experiments. When q is chosen too small, overfitting



(a) $A_0 = 10^{-3}I$. **(b)** $A_0 = I$.

Figure 7-1: Learned value function and typical trajectory (white dots) by VGBP on the inverted pendulum swing-up task with a typical different A_0.

occurs. The temporal difference error becomes very small at the cost of assigning very large values to unvisited states. There are two cases:

- RBF's with centers in unexplored parts of the state space get assigned very positive values. Because these states seem very attractive to the agent, it tries to visit these parts of the state space when possible. Then, the agent gets less return than expected and the overfitted values are corrected.
- Parts of the state space that are assigned a very low value are not visited anymore, because the agent reasons that it is very bad to go there. Hence the wrongly fitted values are not corrected.

The latter case can have a very negative effect on the learning behavior. To give an example the A-matrix is initialized with $A_0 = 10^{-3}I$. The resulting value function is shown in Figure 7-1a. The white dots are the visited states. At both sides of the $[0 \quad 0]^T$ state are valleys in the value function (the two blue c-shaped curves), preventing the agent to apply the extra input needed for the swing-up. In Figure 7-1b the same plot is made for $A_0 = I$. Here the agent has learned to swing the pendulum upright.
To see the influence of the choice of A_0, the learning behavior of different initial matrices is tested. The results are shown in Figure 7-2. A too low initialization leads to overfitting, while a too big value leads to slow learning. The slow learning is caused by the fact that the initialization A_0 is much bigger than the information added by the update (6-34). It therefore takes a long time before the bias in θ is removed.

7-1-2 Eligibility Traces

In this section the influence of the choice of the trace decay factor λ is studied. The learning behavior of VGBP-LSTD(λ) is depicted in Figure 7-3 and 7-4. As can be seen a higher eligibility trace factor leads to faster initial learning and above a certain value to lower maximum performance. The optimum trace decay factor is approximately $\lambda \approx 0.3$. The trace decay

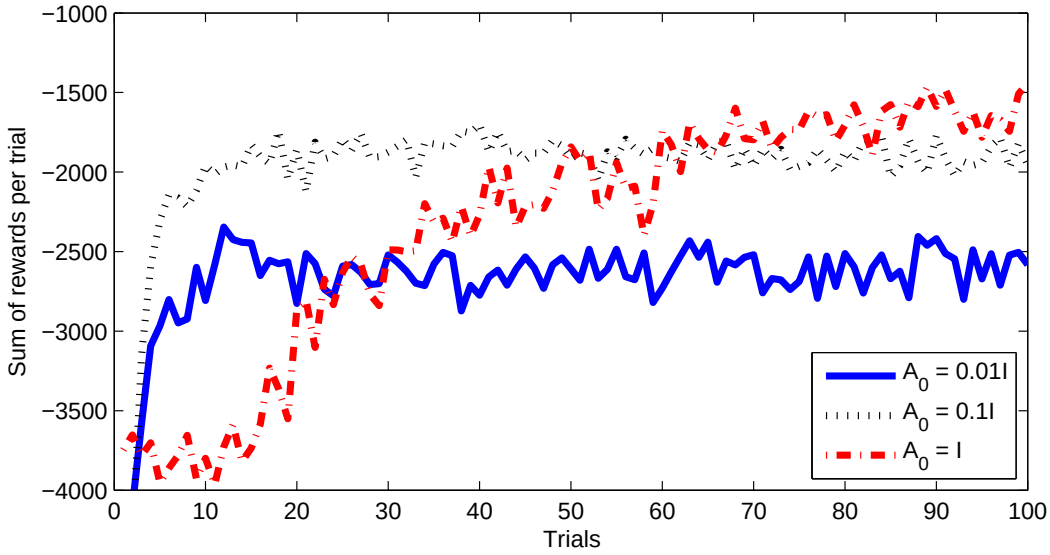


Figure 7-2: Learning behavior of VGBP on the inverted pendulum swing-up task for different choices of A_0.

factor diminishes the influence of the initial A-matrix more quickly. This can be seen from the update equation of the A-matrix:

$$e_{k+1} = \lambda e_k + \phi_c(s_k), \tag{7-1}$$

$$A_{k+1} = A_k + e_{k+1}(\phi_c(s_k) - \gamma\phi_c(s_{k+1}))^T. \tag{7-2}$$

where $e_0 = 0$. Because for $\lambda > 0$ and RBFs $e_k \geq \phi_c(s_k)$ (Equation (7-1)), the outer product added to A at each update step is also bigger. Therefore the influence of the initialization A_0 shrinks more quickly for a bigger trace decay factor. Hence a large trace decay factor also leads to over fitting of the LSTD-critic. This explains the decreased performance for $\lambda = 0.85$.

7-2 Goalkeeper

The experiments with the goalkeeper were briefly discussed in the paper. In this section a more elaborate discussion on the research with the goalkeeper is given. The focus of this chapter lies on the implementation of VGBP on the setup. First the limitations of the setup are discussed. Thereafter the problems faced are discussed. These are split into the curse of dimensionality, the violation of assumptions made in the action selection of VGBP and sensor errors.

7-2-1 System

The joints are actuated by two Dynamixel servos. The servos are position controlled. Using software the system can be made to behave as if the input is a torque signal. These position

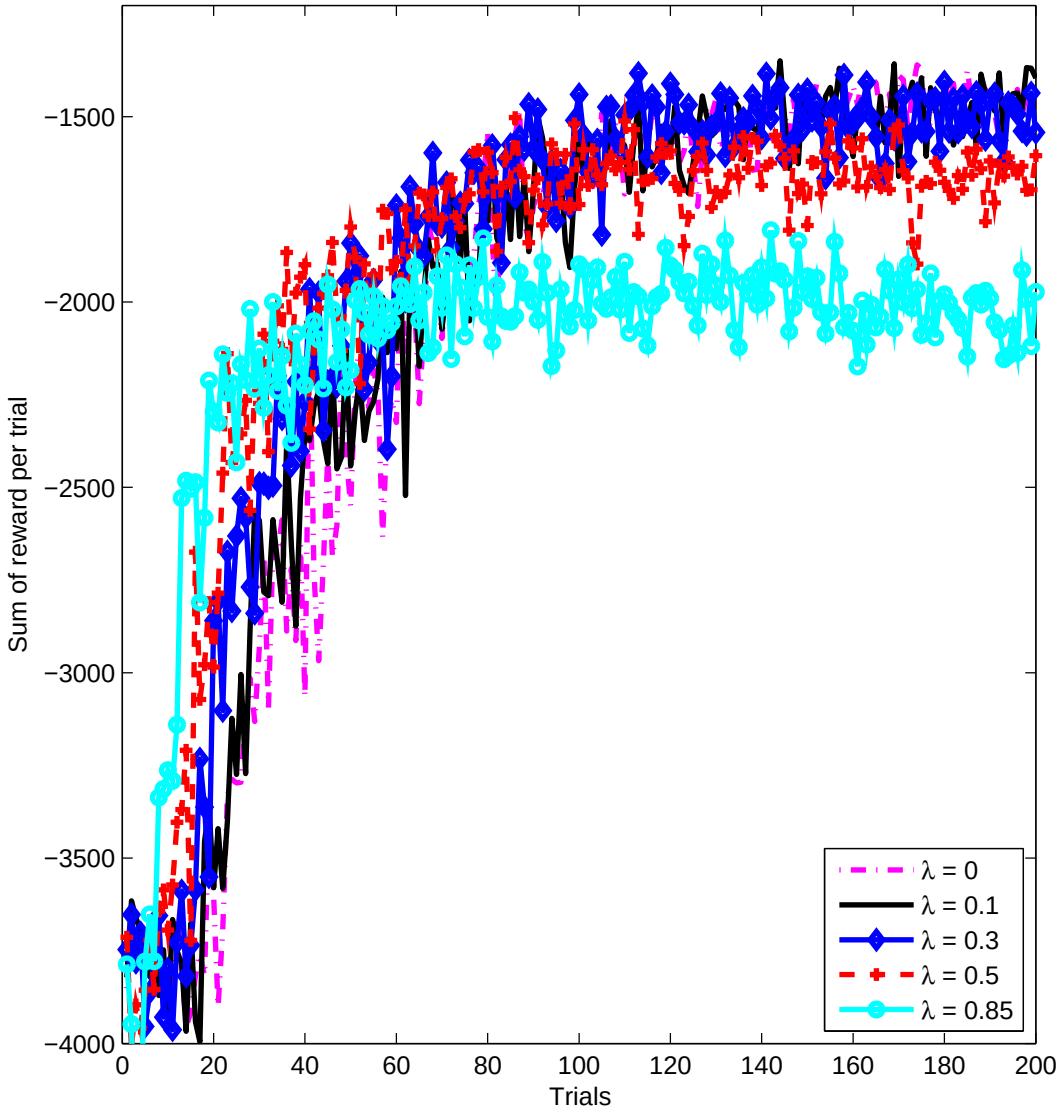


Figure 7-3: Learning behavior of VGBP-LSTD(λ) on simulations of the inverted pendulum swing-up task for different trace decay factors λ.

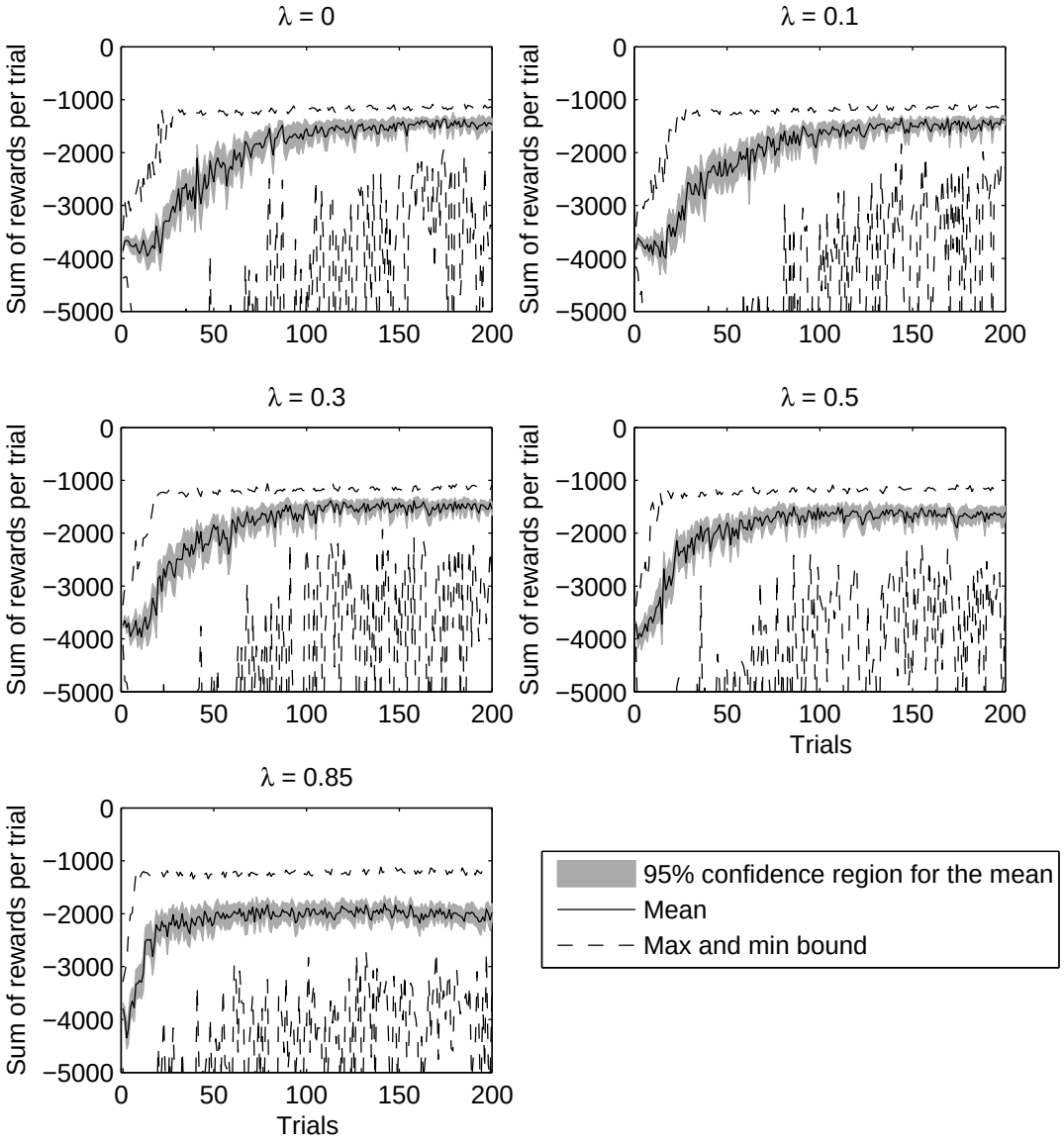


Figure 7-4: Confidence bounds of VGBP-LSTD(λ) on simulations of the inverted pendulum swing-up task for different trace decay factors λ.

input mode and the torque input mode are respectively the *joint mode* and the *wheel mode*. It is possible to switch between these modes on-line. The Dynamixels have a number of difficulties:

- There is a lot of friction on the joints. This violates the assumption of VGBP that the system is linear in the action.
- The emulation of the torque input is not very good. For instance for a zero torque signal in wheel mode the system is much stiffer than it is without power supply.
- The sensor read time is 2 ms per Dynamixel and the write time for both servos together is 1 ms. This gives a minimum sampling time of 5 ms. Due to the high friction and the bad joint mode the dominant system dynamics have a shorter time constant.
- The angle sensor has a limited accuracy of approximately 1°. Furthermore the sensor sometimes gives faulty readings. These errors are at some points even bigger than one radian.
- The angular velocity sensor is inaccurate, hence the angular velocity needs to be obtained by finite difference methods or with an observer.
- The driver sometimes crashes.
- There is backlash in the links.

In all experiments the joint mode is used to put the arm at its initial position. Before the start of a trial the switch is made to wheel mode to enable the agent to control the torques. Both joints have a limited reach. To prevent the system from destroying itself, the state space is limited for both joints to be in $[-1.2, 1.2]$ rad. When the arm is steered outside the allowed state space it needs to be steered back inside the feasible region. A number of braking techniques have been tried. The most effective manner is to switch to joint mode and let the internal controllers move the arm to its center position. Tests have shown that the applied control action by the Dynamixels is higher in joint mode than in wheel mode. Another advantage is that the braking action can be combined with bringing the arm to the start position of the next trial.

7-2-2 Curse of Dimensionality

The quality of the policy is dependent on the precision with which the value function can be represented. In this thesis only equidistant RBF's are used. The center positions are placed on an equidistant grid along the four dimensions of the state space (two angles and two angular velocities). Different types of grids were used. The finest grid that could be implemented was a $7 \times 7 \times 7 \times 7$ grid. This gives 2401 BFs. The limiting factor is the memory needed for the computation of the inverse of the A-matrix. To keep the sampling time within reasonable bounds two modifications had to be made:

- The calculation of the RBF's needs to be sped up. Because the grid of the RBF centers is orthogonal in the dimensions and the scaling matrix is the diagonal matrix

$B = \text{diag}[b_1, \ldots, b_n]$, the calculation of a non-normalized BF can be rewritten as :

$$\bar{\phi}_i(s) = \exp\left(-\frac{1}{2}(s - c_i)^T B^{-1}(s - c_i)\right) \tag{7-3}$$

$$= \exp\left(-\frac{(s(1) - c_i(1))^2}{2b_1}\right) \cdot \ldots \cdot \exp\left(-\frac{(s(n) - c_i(n))^2}{2b_n}\right). \tag{7-4}$$

Because of the orthogonal grid, multiple BFs have the same center along a dimension, e.g. for a 2×2 grid with coordinates:

$$c_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad c_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad c_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad c_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

$c_1(1) = c_2(1)$, and $c_1(2) = c_3(2)$. Therefore the exponentials have to be computed only once for the centers along each dimension. In this way the number of exponentials that needs to be computed for the $7 \times 7 \times 7 \times 7$ grid is reduced from 7^4 to $4 \times 7 = 28$. This is implemented in the function `rbfDiag`. The function reduces the computation time for $\phi(s)$ and $\frac{\partial \phi(s)}{\partial s}$ by approximately 70-90%.

- The update of the A-matrix is taken out of the real time loop. The computation of the outer product $e_{k+1}(\phi(s_k) - \gamma\phi_c(s_{k+1}))^T$ is computationally too expensive for large BFs. Therefore the BFs are stored and the A-matrix is formed in a loop after the trial is completed.[1]

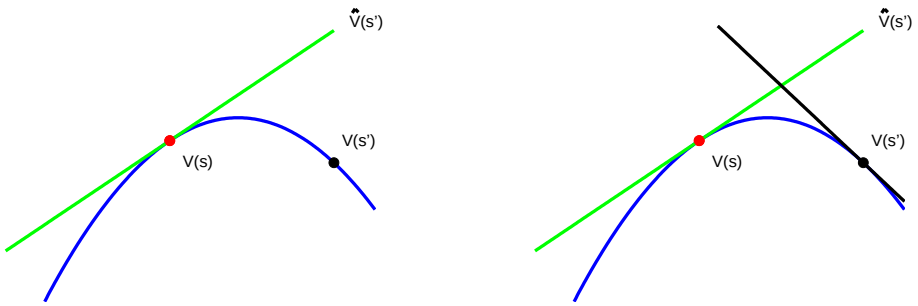
The large number of RBFs makes learning slow. A lot of samples need to be collected to learn a proper policy. This poses a problem: some parts of the state space are very frequently visited (for instance the center position) and some parts of the state space remain unexplored. Therefore the cumulative output of BFs with centers in unexplored parts of the state space will be approximately zero, while the cumulative output of BFs with center positions close to the center position of the arm will be very big. This causes the A-matrix of the critic to become ill conditioned. For the $7 \times 7 \times 7 \times 7$ grid the A-matrix gets ill conditioned before a proper solution is learned. The learned value function and therefore the policy deteriorates. After trying a number of different grid configurations the $5 \times 3 \times 5 \times 3$ grid proved to be the biggest grid that did not suffer from this phenomenon. Therefore this grid is used in the experiments.

7-2-3 Nonlinearity

For VGBP to work properly the system must be linear in the actions and the linear value function approximation must be close to the true value function over the action space. Both assumptions are violated on the Goalkeeper. This is the topic of the two following paragraphs.

Nonlinearity in the Actions The high friction and the backlash in the joints make the system highly nonlinear in the action. This violates the assumption made for the action selection. One possible manner to deal with the nonlinearity in the action is to divide the action space into regions $\mathcal{A}_1, \ldots, \mathcal{A}_n \in \mathcal{A}$ in which the system behaves approximately linearly. An optimal action can be selected in each region. The optimal action over the whole action space is then found by comparing the action values of the optimal actions of all the regions.

[1] This is only done for the tests with a large set of BF's. For tests with smaller sets it is computed online.



(a) No overshoot prevention. **(b)** Overshoot detection.

Figure 7-5: VGBP overshoots the maximum of the value function (in blue) if the linearization (green) is inaccurate over the action space.

Linear Value Function Approximation The value function has a peak near the center position of the robot arm. The slopes towards this peak are very steep. Near the peak the curvature is very large. The sampling time is too long for the linear approximation of the value function to be valid. Because the control action is cheap, the controller favors maximum actions. This results in a bang-bang controller around the center position. This is shown in Figure 7-5a. The agent thinks that the new state has a value $\hat{V}(s')$, but the actual value $V(s')$ is much lower. This phenomenon is clearly visible in the learned trajectory. Figure 7-6 displays a learned trajectory on the Goalkeeper along with the control actions. Possible solutions to this problem could be

- Reducing the sampling time. However, this is not possible on the true system due to the computation time and the reading and writing time of the servos.
- Tuning the reward function. By making the control action more expensive the agent will prefer smaller actions. A disadvantage is that the whole value function is changed. This leads to different behavior. A number of different parameters for the reward function have been tried, but none reduced the bang-bang behavior of the controller.
- The controller can by itself check if the optimum is overshot by the control action. This can be done in the following manner:

  1. Determine the gradient $\frac{\partial V(s)}{\partial a}$ and the control action a for state s.
  2. Predict the next state $\hat{s}'$.
  3. Determine $\frac{\partial V(\hat{s}')}{\partial a}$.
  4. If $\frac{\partial V(\hat{s}')}{\partial a} \cdot \frac{\partial V(s)}{\partial a} > 0$ there is probably no maximum between s and $\hat{s}'$. The control action a can be applied.
     If $\frac{\partial V(\hat{s}')}{\partial a} \cdot \frac{\partial V(s)}{\partial a} < 0$ the maximum is probably overshot. This is shown in Figure 7-5b. The state that has the highest value has to be found by interpolation. For instance the method of false position (Luenberger and Ye, 2008) can be used to find the optimal state. The control action needed to go to this desired state can be determined by an inverse process model. The usage of an inverse process model in action selection is discussed in (Grondman et al., 2011b).

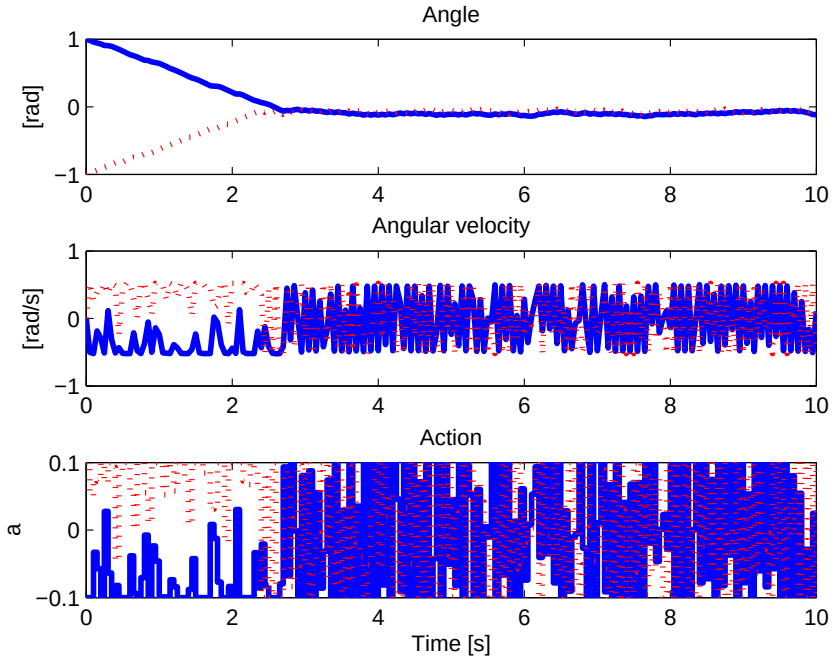


Figure 7-6: Typical learned trajectory by VGBP on simulations of the Goalkeeper, with link 1 (blue solid) and link 2 (red dotted line)

This is a topic for future research.

7-2-4 Sensor Errors

As discussed in Section 7-2-1 the sensors of the goalkeeper occasionally give faulty readings. These errors occur most frequently for the first link in the center position. The errors are sometimes more than a radian, meaning that the trial is aborted and the agent receives the terminal reward. The agent thinks that it is punished for keeping the arm at the center position. The result is that the agent learned to put the first link slightly off center. After this was noticed a linear observer has been implemented. The model for the observer is identified with MOESP (Verhaegen and Verdult, 2007). During the following experiments the sensor errors not only became more frequent, but also lasted for a number of consecutive samples. The observer can not correct these extensive mistakes. Due to time constraints unfortunately no further test with the setup has been performed. The results presented in this thesis are generated with a `Simulink` model.

Chapter 8

Comparison of Value-Gradient Based Policy with Vanilla Actor-Critic

There are a number of similarities between the two introduced algorithms in this thesis. Both Value-Gradient Based Policy (VGBP) and Vanilla Actor-Critic (VAC) make use of the policy gradient. On the inverted pendulum, both algorithms have shown to work best with an LSTD critic. The differences of the two algorithms with an LSTD critic are discussed in this chapter, because these versions are the most thoroughly studied. First the differences in the initialization of the two algorithms are discussed. Thereafter the learning behavior of both algorithms on the inverted pendulum is compared.

8-1 Initialization

The main focus of this thesis is to devise a reinforcement learning (RL) method that is not reliant on parameters that are problem dependent. In this section the dependence of the algorithms on the initial parameters is discussed for the inverted pendulum. The selection concerns the agent parameters excluding the basis functions. The parameters that have to be set can be divided parameters in the critic function and parameters used in the action selection.

Critic Function

The parameters that have to be set for both algorithms are the initial parameters θ_0, A_0, b_0, and the trace decay factor λ and for VAC the forgetting factor η_c.

- The critic parameter vector θ_0. In both algorithms this parameter is initialized as $\theta_0 = 0$. For VAC this parameter has little influence, because it is completely replaced at the first critic update. In VGBP, θ_0 determines the policy until the first update.

- The matrix A_0. This matrix has to be chosen to guarantee invertibility of the A-matrix throughout the learning experiment. It can be used together with b_0 to incorporate a bias in the critic parameter vector update in the early stage of learning (see Section 7-1-1). In this thesis the initial matrix is chosen as $A_0 = qI$. For VAC the constant q has little influence on the learning behavior as long it is chosen in the interval $[10^{-4}, 10^{-2}]$. It is not big enough to introduce a bias in θ, but it is big enough to maintain a proper scaling of the A-matrix.
  For VGBP the selection of A_0 has a drastic influence on the learning behavior. This is thoroughly discussed in Section 7-1-1.

- The vector b_0. With the initial A-matrix $A_0 = qI$, the vector b_0 determines a bias in the parameter vector towards $\theta = b_0$. It can therefore be used together with A_0 to incorporate a priori knowledge. In this thesis it is chosen as $b_0 = 0$, which gives θ a bias towards 0. For VAC $b_0 = 0$ is the most convenient choice because this minimizes the influence of the initialization. For VGBP the bias stimulates exploration. Although not explored in this thesis it could also be used to avoid certain states by assigning very negative values to certain entries.

- The trace decay factor λ. For VGBP the trace decay factor should not be set too large, because the performance deteriorates for too big values. Setting it to $\lambda = 0$ is safe and close to the optimal value. For VAC a value between 0.6 and 0.85 is both safe and reasonably close to the optimal λ. In this range the learning speed is not heavily influenced.

- The forgetting factor η_c. This parameter is only used in VAC. The parameter is introduced to incorporate the non-stationarity of the true value function into LSTD. The optimal value of η_c is determined by the frequency and the magnitude of the policy update. Simulation results have shown that a value close to unity is best. In the interval $[0.90, 0.99]$ performance varies slightly. The parameter could also be introduced in VGBP, but this is left for future research.

For VAC all the critic parameters can be easily picked to fit into a safe set where the learning behavior is nearly optimal. The initialization of the critic of VGBP is very dependent on the choice of A_0. A large initial matrix is safe, because it minimizes overfitting and enhances exploration. A too large initialization on the other hand makes learning quite slow. Therefore a balance needs to be found.

Action Selection

The two algorithms have different parameters and are therefore discussed separately.
VAC has four actor parameters: the initial policy parameter θ_0, the learning rate α_a, the Fisher Information Matrix (FIM) forgetting factor η_F and the maximum difference in consecutive gradient estimates for which an update is performed ϵ.

- The policy parameter vector θ_0 is initialized at zero (in the middle of the action space). Experiments with random initializations also showed good performance, albeit the variance of the learning process increased.

- The actor learning rate α_a has a large influence on the learning behavior. No Hessian estimate was found that could be used for a Levenberg Marquardt (LM) step size. Also the largest eigenvalue could not be used as the learning rate. Therefore the learning rate is found by extensive tuning.
- The forgetting factor η_F has a modest influence when it is set smaller than 0.9.
- The difference in angle ϵ. Because the natural gradient converges quickly the difference in angle can be picked very small. A value of 1° worked well in practice.

The actor parameter of VAC that needed the most tuning on the inverted pendulum is the learning rate α_a. To apply VAC to multidimensional problems, a new estimation of the FIM needs to be made, since it is an integral over the action space.

The parameters used for the action selection in VGBP are in the approximator of the process model. These are the input scaling matrix W, the number of nearest neighbors K and the memory size N.

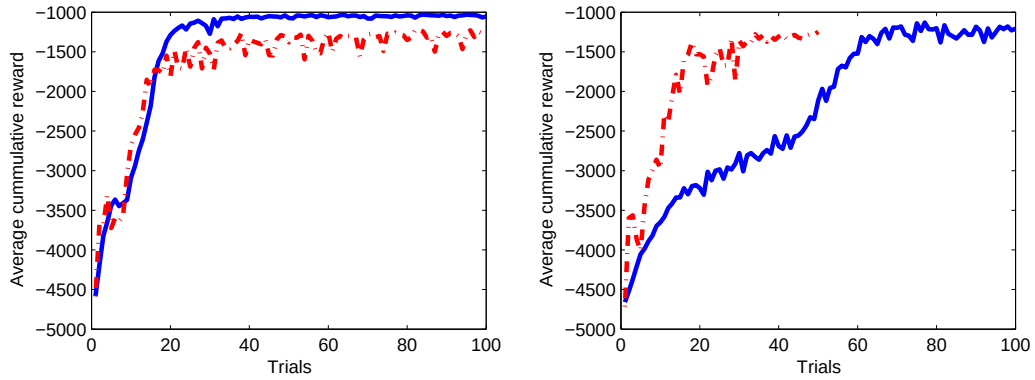
- The weighting matrix W has a big influence on the quality of the model. The entries can be set with prior knowledge about the system.
- The number of nearest neighbors K determines how broad the generalization is. Like the input weighting, some prior knowledge about the system helped to select this parameter. Thereafter very little tuning was required.
- The memory has to contain enough samples to capture the behavior of the system. The value of N is limited by the memory size as well as the computational budget allocated to find the nearest neighbor samples.

Besides the parameters, a solution method to find the optimal action needs to be provided to the agent. For quadratic reward functions VGBP can be easily implemented to different problems with different action dimensions. For more complex reward functions first an analytic action selection method has to be sought. If this is not possible to provide an analytic solution, numerical methods can be used to select the optimal action (if feasible in the given computational budget). Otherwise the solution has to be roughly approximated.

Conclusion On the inverted pendulum the main parameter for VAC to be tuned is the actor learning rate. The learning behavior of VGBP is influenced mostly by the initial matrix A_0 and the LLR parameters W and K. Test on more setups have to be performed to see if these results apply to different problems.

8-2 Learning Behavior

In this section the learning behavior of VAC and VGBP on the inverted pendulum is compared. The average learning curves for both algorithms are given in Figure 8-1. In simulation the convergence speed of the algorithms is very similar. The main difference is the return of



(a) In simulation. **(b)** On the setup.

Figure 8-1: Average learning behavior of VAC (blue solid) and VGBP (red dash-dotted) on the pendulum swing-up task.

the final policy. Where VAC always needs only two swings to get the pendulum to the top position, VGBP takes two or three swings to get the pendulum upright. This is also the reason that the variance of the final policy is smaller for VAC.
The learning behavior on the setup is shown in Figure 8-1b. The learning experiments for VGBP lasted 50 trials, which with a trial length of 3 seconds corresponds to 2.5 minutes of learning, because the algorithm converges in 30 trials. The following points can be noticed:

- The quality of the final policies is almost equal. Both algorithms learn to get the pendulum upright in two swings. VGBP takes less swings on the setup than in simulation. The final performance is probably less than the best performance on the setup due to the simplistic model used in simulation.
- The convergence speed of VGBP in simulation and on the the setup is very similar. In contrast, the convergence speeds of VAC is very affected by the transition to the setup, because the quality of the policy gradient decreased.

VAC seems to find the optimal policy both in simulation and on the setup. VGBP on the other hand may find a suboptimal policy, but the learning speed is high and not much influenced by the transition to the actual system.

Chapter 9

Conclusion and Future Work

The main goal of this thesis was to reduce dependence of reinforcement learning (RL) on problem dependent parameters. The main results of the thesis are given in this chapter along with recommendations for future work.

9-1 Summary

In this thesis two different approaches have been taken to remove the dependence of the action selection on learning parameters. In Chapter 3 to 5 it was investigated if the Levenberg Marquardt (LM) algorithm could be used to select the actor step size. The second approach taken was to remove the actor and hence the actor parameters. This was treated in Chapter 6 and 7. The findings of the two approaches are discussed in separate sections.

9-1-1 Policy Gradient Actor-Critic Algorithms

The most important parameter in the (actor) update of policy gradient algorithms is the step size. To determine the step size automatically with the LM algorithm the vanilla policy gradient and the policy Hessian are required. Therefore the Vanilla Actor-Critic (VAC) algorithm has been devised. The vanilla gradient is obtained by multiplying the natural gradient with the Fisher Information Matrix (FIM). In Chapter 5 methods are investigated to obtain the policy Hessian. The most important conclusions about the policy gradient actor-critic methods are:

- The estimation of the policy Hessian is not straightforward. A number of Hessian estimation techniques have been tested. The Hessian approximations suffer from the following problems:
  - The approximation is not negative definite near an optimum.

  - The approximation does not converge quickly enough to be practically implementable on an RL algorithm. The Broydon-Fletcher-Goldfarb-Shanno (BFGS) method does not converge at all.
  - The Hessian is badly conditioned. The Hessian is not numerically invertible, even for policy vectors close to a (local) optimum.

  Probable causes of the bad Hessian approximation are the variance in the gradient estimates and the sparse structure of the actor basis functions. Because no proper Hessian estimate was found, an LM actor update method could not be constructed.

- The devised VAC algorithm outperforms NAC on the inverted pendulum swing-up task both in simulation and on the physical system. NAC is more susceptible to overfitted values of the natural gradient introduced by Least-Squares Temporal Difference (LSTD). In the VAC update the overfitted values of the natural gradient are removed by the multiplication with the FIM. For a TD-critic the natural gradient algorithm learns faster than its vanilla gradient counterpart.
- On the inverted pendulum the actor learning rate is the parameter that needs the most fine tuning for both NAC and VAC. The other learning parameters have a broader range in which the learning behavior is near optimal.
- The learning behavior of both VAC and NAC is much worse on physical setup than it is in simulation. The cause is the decreased quality of the policy gradient used in the actor update.

9-1-2 Value-Gradient Based Policy

Value-Gradient Based Policy (VGBP) selects actions by optimization of the right hand side of the Bellman equation. For this optimization the agent needs know the consequences of its actions. These consist of the reward and the state transition. The reward received for every state action pair is known by by the agent, because it has direct access to the reward function. In this thesis VGBP learns the process model online with local linear regression (LLR). The most important conclusions about the algorithm are:

- The algorithm achieves a very high learning speed, especially with an LSTD critic. The learning speed on the true system is very comparable to the learning speed in simulation for the pendulum.
- The algorithm is built around the assumptions that the value function can be approximated locally with an affine model and that the system dynamics are linear in the action. When these assumptions are violated, for instance by a too long sampling time, the algorithm gets a tendency to overshoot the optimum. This results in a bang-bang controller in the vicinity of an optimum in the value function (like the center position of for the Goalkeeper experiment).
- The function that finds the optimal action has to be designed before the learning experiment. For a convex reward function there is a single optimal action. For a quadratic reward function the optimal action can be easily calculated. For more complex reward structures it is harder to devise a function that has the optimal action as output.

- The learning parameters that are hardest to determine for the inverted pendulum swing-up task, are the initial matrix A_0 for the LSTD critic and the weighting matrix W and the number of nearest neighbors K for the process model.

9-2 Recomendations and Future Work

Based on the afore mentioned conclusions, the following recommendations can be made:

- The learning behavior of VGBP could benefit from a smart initialization of the value function based on information contained in the reward function. This could be used to let the agent avoid bad states. For instance for a mechanical system, actions that could damage the system could be avoided. This would allow to safely apply VGBP to mechanical systems with physical limitations.
- The method to estimate the policy Hessian very recently introduced by Furmston and Barber (2012), can possibly be extended to the infinite horizon actor-critic scheme. This would entail that no actor learning rate needs to be tuned. In combination with an LSTD critic, a learning rate-free actor-critic method could be devised.
- The performance of VGBP-LSTD is very dependent on the initialization of the A-matrix. For a too small initialization, overfitting of the critic reduces exploration, which leads to suboptimal policies. If it is possible to automatically determine if overfitting occurs, the learning experiment could be restarted with a bigger initial matrix A_0. The algorithm is then independent on the critic initialization.
- The quality of the policy of VGBP is dependent on the accuracy of the learned process model. For the LLR approximator the most influential parameters (the weighting matrix W and the number of nearest neighbors K) are now prespecified. If these parameters could be tuned automatically the VGBP is less dependent on its initial settings.
- To gain full insight in the difference between the natural policy gradient and the introduced vanilla gradient estimate, experiments need to be conducted for different parameterizations on a number of benchmark problems. For a fair comparison both vanilla and natural gradient algorithms should use the same actor and critic function approximators.
- The Hessian estimation techniques were tested with a policy that is linear in the parameters with a high dimensional basis function on a complex problem. It is worth to investigate if these methods work for smaller parameterizations and simpler problems. This could provide more insight in the advantages and limitations of Hessian estimation techniques applied to RL.

The presented work provides insights in the dependence of continuous RL algorithms on learning parameters. It further more shows the difficulties that are involved with reducing the dependence on these parameters. If the above mentioned issues are solved, important steps could be taken towards the construction of RL algorithms that achieve quick learning on practically any control problem, without the need to tune problem dependent learning parameters.

Appendix A

Mathematical Derivations

A-1 Gauss Policy

A Gaussian policy selects actions according to:

$$a \sim \mathcal{N}(\mu, \sigma^2), \tag{A-1}$$

were μ is the mean determined by the policy parameter ϑ and σ is the predetermined standard deviation. The probability density function is given by

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(a-\mu)^2}{2\sigma^2}\right). \tag{A-2}$$

In this thesis the mean of the policy π for a state s is determined with $\mu = \vartheta^T \phi(s)$, using the policy parameter ϑ and the feature vector $\phi(s)$.[1] The derivations in this appendix will be done for an one dimensional action.

A-1-1 Advantage Feature Function for a Gaussian Policy

As given by the Policy Gradient Theorem (Sutton et al., 1999), the feature vector of the advantage function is $\frac{\partial \log \pi(s,a)}{\partial \vartheta}$. Because the chance of selecting a particular action is zero ($\pi(s,a) = 0 \quad \forall a \in \mathcal{A}$), the probability density function is used to derive the feature function:

$$\frac{\partial \log \pi(s,a)}{\partial \vartheta} = \frac{\partial}{\partial \vartheta} \log\left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(a-\mu)^2}{2\sigma^2}\right)\right) \tag{A-3}$$

$$= \frac{\partial}{\partial \vartheta}\left(\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{(a-\mu)^2}{2\sigma^2}\right) \tag{A-4}$$

$$= \frac{a-\mu}{\sigma^2}\frac{\partial \mu(s)}{\partial \vartheta}. \tag{A-5}$$

[1] The subscript a is dropped, because only the actor BF is considered in the appendix.

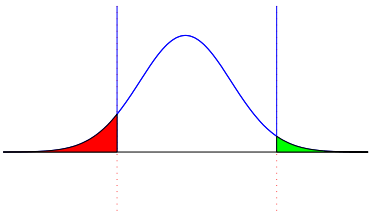


Figure A-1: Gaussian policy for an agent where the saturation of the action space is part of the agent.

For an approximation that is linear in the parameters (LIP) this becomes:

$$\frac{\partial \log \pi(s,a)}{\partial \vartheta} = \frac{a-\mu}{\sigma^2}\phi(s). \tag{A-6}$$

A-1-2 Policy Gradient for a Gaussian Policy with Linearly Approximated Mean

The gradient of the return ρ with respect to the policy parameter ϑ is given by the Policy Gradient Theorem with function approximation (Peters and Schaal, 2008a):

$$\frac{\partial \rho}{\partial \vartheta} = \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \pi(s,a) \frac{\partial \log \pi(s,a)}{\partial \vartheta} \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T \mathrm{d}a\mathrm{d}s \cdot w \tag{A-7}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \int_{\mathcal{A}} \pi(s,a) \frac{\partial \log \pi(s,a)}{\partial \vartheta} \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T \mathrm{d}a \cdot w, \tag{A-8}$$

where d^{π} is the steady state distribution and w is the advantage function parameter. The integral in (A-8) is over the complete action space. If the agent has a limited action space there are two options:

- The saturation is part of the environment. The action space is $(-\infty, \infty)$.
- The limitation is incorporated in the agent. The integral of the policy over the probability space is cut into three parts: a part smaller than the minimum bound, the feasible action space, and a part bigger than the maximum bound. This is shown in Figure A-1. The chance of selecting the minimum action is given by the red area, and the chance of selection the maximum action is the green area. To solve the integral over the action space of Equation (A-8) an analytic expression for the probability of selecting the maximum and the minimum action is required. For the maximum action this is:

$$\pi(s, a_{\max}) = \frac{1}{\sqrt{2\pi}\sigma} \int_{a_{\max}}^{\infty} \exp\left(-\frac{(a-\mu)^2}{2\sigma 2}\right) \mathrm{d}a. \tag{A-9}$$

  Because there is no explicit expression for $\pi(s, a_{\max})$ and $\pi(s, a_{\min})$ (Dekking et al., 2010), this option is infeasible.

With the action space set to $(-\infty, \infty)$, the integral in Equation A-8 is solved by substitution of A-6. For simplicity $\bar{a} = a - \mu$ is used. Thus,

$$\int_{\mathcal{A}} \pi(s,a) \frac{\partial \log \pi(s,a)}{\partial \vartheta} \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T \mathrm{d}a = \int_{\bar{\mathcal{A}}} \pi(s,a) \frac{\bar{a}^2}{\sigma^4} \phi(s)\phi(s)^T d\bar{a} \tag{A-10}$$

$$= \int_{\bar{\mathcal{A}}} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\bar{a}^2}{2\sigma^2}\right) \bar{a}^2 d\bar{a} \frac{\phi(s)\phi(s)^T}{\sigma^4} \tag{A-11}$$

$$= \int_{\bar{\mathcal{A}}} \exp\left(\frac{-\bar{a}^2}{2\sigma^2}\right) \bar{a}^2 d\bar{a} \frac{\phi(s)\phi(s)^T}{\sqrt{2\pi}\sigma^5} \tag{A-12}$$

$$= \sigma^3 \sqrt{2\pi} \frac{\phi(s)\phi(s)^T}{\sqrt{2\pi}\sigma^5} \tag{A-13}$$

$$= \frac{\phi(s)\phi(s)^T}{\sigma^2}. \tag{A-14}$$

Substituting this result in Equation (A-8) gives the unbiased estimator:

$$\frac{\partial \rho}{\partial \vartheta} \approx \frac{1}{\sigma^2 N} \sum_{i=1}^{N} \phi(s_i)\phi(s_i)^T w. \tag{A-15}$$

A-1-3 Derivatives of a Gaussian Policy w.r.t. ϑ

$$\frac{\partial \pi(s,a)}{\partial \vartheta} = \frac{a - \mu(s)}{\sigma^3 \sqrt{2\pi}} \exp\left(-\frac{(a-\mu(s))^2}{2\sigma^2}\right) \frac{\partial \mu(s)}{\partial \vartheta} \tag{A-16}$$

$$\frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} = \frac{1}{\sigma^3 \sqrt{2\pi}} \left((a-\mu(s)) \frac{\partial^2 \mu(s)}{\partial \vartheta^2} + \left(\frac{(a-\mu(s))^2}{\sigma^2} - 1 \right) \frac{\partial \mu(s)}{\partial \vartheta} \frac{\partial \mu(s)}{\partial \vartheta}^T \right) \exp\left(-\frac{(a-\mu(s))^2}{2\sigma^2}\right) \tag{A-17}$$

A-1-4 Performance Hessian of the Gaussian Policy w.r.t ϑ

In this section the Hessian of the performance of the Gauss policy is derived. The Hessian of the performance is given by[2]

$$\frac{\partial^2 \rho}{\partial \vartheta^2} = \sum_s d^\pi(s) \sum_a \left[\frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} Q^\pi(s,a) + \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial Q^\pi(s,a)}{\partial \vartheta}^T + \frac{\partial Q^\pi(s,a)}{\partial \vartheta} \frac{\partial \pi(s,a)}{\partial \vartheta}^T \right]. \tag{A-18}$$

Policy gradient reinforcement learning uses the following parametrization of the Q-function (Sutton et al., 1999):

$$Q(s,a) = b(s) + A(s,a) \tag{A-19}$$

where $b(s)$ is a baseline function, often the value function, and A is an advantage function. Using the advantage function $A = \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T w$, where w is the natural gradient, the first

[2] See Appendix A-2 for the derivation.

part of the right hand side of Equation A-18 becomes:

$$\sum_a \frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} Q^\pi(s,a) = \sum_a \frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} \left(b(s) + \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T w \right) \tag{A-20}$$

$$= \sum_a \frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T w. \tag{A-21}$$

The baseline drops out of the equation because $\sum_a \frac{\partial \pi(s,a)}{\partial \vartheta} = 0 \quad \forall s$. For a Gaussian policy with a mean that is linear in the parameters, the following substitutions can be made (see Section A-1-3):

$$\frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} = \frac{1}{\sigma^5 \sqrt{2\pi}} \left(\bar{a}^2 - \sigma^2 \right) \phi(s) \phi(s)^T \exp\left(-\frac{\bar{a}^2}{2\sigma^2} \right), \tag{A-22}$$

$$\frac{\partial \log \pi(s,a)}{\partial \vartheta} = \frac{\bar{a}}{\sigma^2} \phi(s), \tag{A-23}$$

where $\bar{a} = a - \mu(s)$.The summation has to be changed into an integral:

$$\int_{\mathcal{A}} \frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} Q^\pi(s,a) \mathrm{d}a = \frac{(\phi(s)^T w) \phi(s) \phi(s)^T}{\sigma^7 \sqrt{2\pi}} \int_{\bar{\mathcal{A}}} (\bar{a}^3 - \bar{a}\sigma^2) \exp\left(-\frac{\bar{a}^2}{2\sigma^2} \right) \mathrm{d}\bar{a}. \tag{A-24}$$

Because $\int_{\mathcal{A}} (\bar{a}^3 - \bar{a}\sigma^2) \exp\left(-\frac{\bar{a}^2}{2\sigma^2}\right) \mathrm{d}a$ is an odd function, its integral over the action space $(-\infty, \infty)$ is zero. The Hessian of the performance of a Gaussian policy reduces to:

$$\frac{\partial^2 \rho}{\partial \vartheta^2} = \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial Q^\pi(s,a)}{\partial \vartheta}^T + \frac{\partial Q^\pi(s,a)}{\partial \vartheta} \frac{\partial \pi(s,a)}{\partial \vartheta}^T \mathrm{d}a \, \mathrm{d}s. \tag{A-25}$$

This can be simplified by using the compatible function approximation:

$$\frac{\partial Q^\pi(s,a)}{\partial \vartheta} = \frac{\partial \left(b(s) + \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T w \right)}{\partial \vartheta} \tag{A-26}$$

$$= \frac{\partial b(s)}{\partial \vartheta} + \frac{\partial (\frac{\partial \log \pi(s,a)}{\partial \vartheta}^T w)}{\partial \vartheta}. \tag{A-27}$$

The expression $\frac{\partial b(s)}{\partial \vartheta}$ will only exist if the baseline varies with the policy parameter. Applying the product rule to the right term in (A-27) gives:

$$\frac{\partial (\frac{\partial \log \pi(s,a)}{\partial \vartheta}^T w)}{\partial \vartheta} = \frac{\partial^2 \log \pi(s,a)}{\partial \vartheta}^T w + \frac{\partial w}{\partial \vartheta}^T \frac{\partial \log \pi(s,a)}{\partial \vartheta} \tag{A-28}$$

$$= -\frac{\phi(s)\phi(s)^T}{\sigma^2} w + \frac{\partial w}{\partial \vartheta}^T \frac{a - \mu}{\sigma^2} \phi(s). \tag{A-29}$$

Taking one expression of the integral in Equation A-25 and substituting this result gives:

$$\int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial Q^\pi(s,a)}{\partial \vartheta}^T \mathrm{d}a = \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} \left(\frac{\partial b(s)}{\partial \vartheta} - \frac{\phi(s)\phi(s)^T}{\sigma^2} w \right)^T \mathrm{d}a + \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T \frac{\partial w}{\partial \vartheta} \mathrm{d}a \tag{A-30}$$

$$= \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T \frac{\partial w}{\partial \vartheta} \mathrm{d}a. \tag{A-31}$$

The first expression on the right hand side drops because the derivative of the summation over all actions is zero. Substituting both the derivative of the policy and the compatible feature in (A-31) :

$$\int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial \log \pi(s,a)}{\partial \vartheta}^T \frac{\partial w}{\partial \vartheta} \mathrm{d}a = \frac{\phi(s)\phi(s)^T}{\sigma^5 \sqrt{2\pi}} \int_{\bar{\mathcal{A}}} \bar{a} \exp\left(-\frac{\bar{a}^2}{2\sigma^2}\right) \mathrm{d}\bar{a} \tag{A-32}$$

$$= \frac{1}{\sigma^2} \phi(s)\phi(s)^T \frac{\partial w}{\partial \vartheta}. \tag{A-33}$$

The Hessian of the performance ρ with respect to ϑ for the Gauss policy is thus:

$$\frac{\partial^2 \rho}{\partial \vartheta^2} = F \frac{\partial w}{\partial \vartheta} + \frac{\partial w}{\partial \vartheta}^T F, \tag{A-34}$$

with the Fisher matrix F denoted by:

$$F = \frac{1}{\sigma^2} \int_{\mathcal{S}} d^\pi(s)\phi(s)\phi(s)^T \mathrm{d}s. \tag{A-35}$$

A-2 Hessian of the Return in the Average Reward Setting

The value function $V^\pi(s)$ can be derived from the Q-function $Q(s,a)$ and the policy $\pi(s,a)$ by:

$$V^\pi(s) = \sum_a \pi(s,a) Q^\pi(s,a). \tag{A-36}$$

To find the Hessian of the value of a state with respect to the policy parameter, one of the sides of Equation A-36 needs to be differentiated twice. Applying the chain rule to Equation A-36 gives:

$$\frac{\partial^2 V^\pi(s)}{\partial \vartheta^2} = \frac{\partial^2}{\partial \vartheta^2} \left(\sum_a \pi(s,a) Q^\pi(s,a) \right) \tag{A-37}$$

$$= \frac{\partial}{\partial \vartheta} \left(\sum_a \left[\frac{\partial \pi(s,a)}{\partial \vartheta} Q^\pi(s,a) + \pi(s,a) \frac{\partial Q^\pi(s,a)}{\partial \vartheta} \right] \right) \tag{A-38}$$

$$= \sum_a \left[\frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} Q^\pi(s,a) + \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial Q^\pi(s,a)}{\partial \vartheta}^T + \right.$$

$$\left. \frac{\partial Q^\pi(s,a)}{\partial \vartheta} \frac{\partial \pi(s,a)}{\partial \vartheta}^T + \pi(s,a) \frac{\partial^2 Q^\pi(s,a)}{\partial \vartheta^2} \right]. \tag{A-39}$$

Because the Hessian of the performance ρ is needed, it is more convenient to use the following definition of the Q-function:

$$Q^\pi(s,a) = \mathcal{R}(s,a) - \rho(\pi) + \sum_{s'} \mathcal{P}(s,a,s') V^\pi(s'). \tag{A-40}$$

The second derivative of the Q-function can be rewritten using Equation A-40 as:

$$\frac{\partial^2 Q^\pi(s,a)}{\partial \vartheta^2} = \frac{\partial^2}{\partial \vartheta^2} \left[\mathcal{R}(s,a) - \rho(\pi) + \sum_{s'} \mathcal{P}(s,a,s') V^\pi(s') \right] \tag{A-41}$$

$$= -\frac{\partial^2 \rho}{\partial \vartheta^2} + \sum_{s'} \mathcal{P}(s,a,s') \frac{\partial^2 V^\pi(s')}{\partial \vartheta^2}. \tag{A-42}$$

No (second) derivatives of the reward function $\mathcal{R}(s,a)$ and the transition model $\mathcal{P}(s,a,s')$ occur, because these functions are not differentiable with respect to ϑ. The results can be substituted in Equation A-39, giving:

$$\frac{\partial^2 V^\pi(s)}{\partial \vartheta^2} = \sum_a \left[\frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} Q^\pi(s,a) + \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial Q^\pi(s,a)}{\partial \vartheta}^T + \frac{\partial Q^\pi(s,a)}{\partial \vartheta} \frac{\partial \pi(s,a)}{\partial \vartheta}^T + \pi(s,a) \left(-\frac{\partial^2 \rho}{\partial \vartheta^2} + \sum_{s'} \mathcal{P}(s,a,s') \frac{\partial^2 V^\pi(s')}{\partial \vartheta^2} \right) \right]. \tag{A-43}$$

The goal is to find the Hessian of the performance ρ. Therefore $\frac{\partial^2 \rho}{\partial \vartheta^2}$ needs to be isolated. This is possible because:

$$\sum_a \pi(s,a) \frac{\partial^2 \rho}{\partial \vartheta^2} = \frac{\partial^2 \rho}{\partial \vartheta^2}. \tag{A-44}$$

Isolating $\frac{\partial^2 \rho}{\partial \vartheta^2}$ gives:

$$\frac{\partial^2 \rho}{\partial \vartheta^2} = \sum_a \left[\frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} Q^\pi(s,a) + \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial Q^\pi(s,a)}{\partial \vartheta}^T + \frac{\partial Q^\pi(s,a)}{\partial \vartheta} \frac{\partial \pi(s,a)}{\partial \vartheta}^T + \pi(s,a) \sum_{s'} \mathcal{P}(s,a,s') \frac{\partial^2 V^\pi(s')}{\partial \vartheta^2} \right] - \frac{\partial^2 V^\pi(s)}{\partial \vartheta^2}. \tag{A-45}$$

It is inconvenient that the performance Hessian depends on all possible next states s'. If a summation is made over the steady state distribution $d^\pi(s)$, due to the stationarity the following holds

$$\sum_s d^\pi(s) \sum_a \pi(s,a) \sum_{s'} \mathcal{P}(s,a,s') \frac{\partial^2 V^\pi(s')}{\partial \vartheta^2} = \sum_s d^\pi(s) \sum_a \frac{\partial^2 V^\pi(s)}{\partial \vartheta^2}. \tag{A-46}$$

Summing Equation A-45 over the steady state distribution gives:

$$\begin{aligned} \sum_s d^\pi(s) \frac{\partial^2 \rho}{\partial \vartheta^2} = &\sum_s d^\pi(s) \sum_a \frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} Q^\pi(s,a) + \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial Q^\pi(s,a)}{\partial \vartheta}^T \\ &+ \sum_s d^\pi(s) \sum_a \frac{\partial Q^\pi(s,a)}{\partial \vartheta} \frac{\partial \pi(s,a)}{\partial \vartheta}^T \\ &+ \sum_s d^\pi(s) \sum_a \pi(s,a) \sum_{s'} \mathcal{P}(s,a,s') \frac{\partial^2 V^\pi(s')}{\partial \vartheta^2} - \sum_s d^\pi(s) \sum_a \frac{\partial^2 V^\pi(s)}{\partial \vartheta^2}, \end{aligned} \tag{A-47}$$

which due to the stationarity reduces to:

$$\frac{\partial^2 \rho}{\partial \vartheta^2} = \sum_s d^\pi(s) \sum_a \left[\frac{\partial^2 \pi(s,a)}{\partial \vartheta^2} Q^\pi(s,a) + \frac{\partial \pi(s,a)}{\partial \vartheta} \frac{\partial Q^\pi(s,a)}{\partial \vartheta}^T + \frac{\partial Q^\pi(s,a)}{\partial \vartheta} \frac{\partial \pi(s,a)}{\partial \vartheta}^T \right] \tag{A-48}$$

A-3 An Assumption of an Approximate Hessian

In Section 5-1-3 it is assumed that an approximate Hessian can be of the form:

$$H_2(\vartheta) = -\int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \pi_\vartheta(s,a) Q(s,a) \left[\frac{\partial^2 \log \pi_\vartheta(s,a)}{\partial \vartheta^2} \right] \mathrm{d}a \, \mathrm{d}s \tag{A-49}$$

If this is true the derivation below holds:

The Hessian of the policy with respect to the policy parameter is given by:

$$\frac{\partial^2 \log \pi_\vartheta(s,a)}{\partial \vartheta^2} = \frac{\partial}{\partial \vartheta} \frac{a-\mu}{\sigma^2}\phi(s) = \frac{1}{\sigma^2}\phi(s)\phi(s)^T \tag{A-50}$$

To compute the integral over the action space, the policy probability density function and the compatible Q-function $Q(s,a) = A(s,a) + b(s)$ with the advantage function A and baseline $b(s)$ are used. Substituting these functions in Equation (A-49) gives:

$$\begin{aligned} H_2(\vartheta) = &- \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \pi_\vartheta(s,a) A(s,a) \left[\frac{\partial^2 \log \pi_\vartheta(s,a)}{\partial \vartheta^2} \right] \mathrm{d}a\mathrm{d}s \\ &- \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \pi_\vartheta(s,a) b(s) \left[\frac{\partial^2 \log \pi_\vartheta(s,a)}{\partial \vartheta^2} \right] \mathrm{d}a\mathrm{d}s. \end{aligned} \tag{A-51}$$

The integral containing the advantage function can be calculated by:

$$\int_{\mathcal{A}} \pi_\vartheta(s,a) A(s,a) \left[\frac{\partial^2 \log \pi_\vartheta(s,a)}{\partial \vartheta^2} \right] \mathrm{d}a \tag{A-52}$$

$$= \int_{\mathcal{A}} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(a-\mu)^2}{2\sigma^2} \right) \frac{a-\mu}{\sigma^2} \left(\phi_a(s)^T w \right) \frac{1}{\sigma^2}\phi(s)\phi(s)^T \mathrm{d}a. \tag{A-53}$$

where w is the natural gradient. By using $\bar{a} = a - \mu$ and taking all the constants with respect to $\bar{a}$ out of the integral, this becomes:

$$\frac{1}{\sigma^5\sqrt{2\pi}} \left(\phi_a(s)^T w \right) \phi(s)\phi(s)^T \int_{\bar{\mathcal{A}}} \bar{a} \exp\left(-\frac{(a-\mu)^2}{2\sigma^2} \right) \mathrm{d}\bar{a}. \tag{A-54}$$

This is always zero, because the integral over the action space $(-\infty, \infty)$ is zero for odd functions. The integral of the baseline is:

$$\int_{\mathcal{A}} \pi_\vartheta(s,a) b(s) \left[\frac{\partial^2 \log \pi_\vartheta(s,a)}{\partial \vartheta^2} \right] \mathrm{d}a \tag{A-55}$$

$$= b(s) \frac{1}{\sigma^2}\phi(s)\phi(s)^T \int_{\mathcal{A}} \pi_\vartheta(s,a) \mathrm{d}a \tag{A-56}$$

$$= b(s) \frac{1}{\sigma^2}\phi(s)\phi(s)^T. \tag{A-57}$$

The Hessian H_2 is given by:

$$H_2 = -\frac{1}{\sigma^2} \int_{\mathcal{S}} d^\pi(s) b(s) \phi(s)\phi(s)^T \mathrm{d}s. \tag{A-58}$$

A-4 Derivative of Radial Basis Functions with Respect to the State

Radial basis functions transform a state $s \in \mathbb{R}^n$ into a feature vector $\phi(s) \in \mathbb{R}^m$ that specifies the properties of the state. The i-th element of the feature vector $\phi(s)$ is denoted by $\phi_i(s)$ and is calculated with:

$$\phi_i(s) = \frac{\bar{\phi}_i(s)}{\sum_{i'=1}^n \bar{\phi}_{i'}(s)}, \qquad \bar{\phi}_i(s) = \exp\left(-\frac{1}{2}(s - c_i)^T B_i^{-1} (s - c_i) \right), \tag{A-59}$$

where $\bar{\phi}_i(s)$ is the non normalized feature. The centers of the RBFs are given by the vector $c_i = [c_{i,1}, c_{i,2}, \ldots, c_{i,D}]^T \in \mathbb{R}^D$ and the width is determined by the symmetric positive definite matrix $B \in \mathbb{R}^{D \times D}$. The derivative of an element of the basis function with respect to the j-th element of the state s_j is:

$$\frac{\partial \phi_i(s)}{\partial s_j} = \left(\frac{\partial}{\partial s_j} \frac{1}{\sum_{i'=1}^{n} \bar{\phi}_{i'}(s)} \right) \bar{\phi}_i(s) + \frac{1}{\sum_{i'=1}^{n} \bar{\phi}_{i'}(s)} \frac{\partial \bar{\phi}_i(s)}{\partial s_j} \ . \tag{A-60}$$

Where the product rule is used. In case the scaling matrix B_i^{-1} is equal for all BF's and B is a diagonal matrix given by $B^{-1} = \text{diag}(b_1, \ldots, b_n)$, the derivative of $\bar{\phi}_i$ can be found by applying the chain rule to Equation A-4:

$$\frac{\partial \bar{\phi}_i(s)}{\partial s_j} = -b_j(s_j - c_{i,j})\bar{\phi}_i(s). \tag{A-61}$$

The derivative of the normalizing term is:

$$\frac{\partial}{\partial s_j} \frac{1}{\sum_{i'=1}^{n} \bar{\phi}_{i'}(s)} = -\frac{\sum_{i'=1}^{n} \frac{\partial \bar{\phi}_{i'}(s)}{\partial s_j}}{(\sum_{i'=1}^{n} \bar{\phi}_{i'}(s))^2} \tag{A-62}$$

$$= \frac{\sum_{i'=1}^{n} b_j(s_j - c_{i',j})\bar{\phi}_{i'}(s)}{(\sum_{i'=1}^{n} \bar{\phi}_{i'}(s))^2}. \tag{A-63}$$

By combining Equation A-61 and A-63, the final result is:

$$\frac{\partial \phi_i(s)}{\partial s_j} = \frac{\sum_{i'=1}^{n} \left(b_j(s_j - c_{i',j})\bar{\phi}_{i'}(s) \right) - b_j(s_j - c_{i,j})\phi_{\text{sum}}(s)}{\phi_{\text{sum}}^2} \bar{\phi}_i(s), \tag{A-64}$$

with

$$\phi_{\text{sum}}(s) = \sum_{i'=1}^{n} \bar{\phi}_{i'}(s). \tag{A-65}$$

Appendix B

Overview Actor-Critic toolbox

In this Appendix an overview is given of all the actor-critic algorithms and functions written in the toolbox. For each function it is mentioned for which chapter it is used. The main directory has three main folders:

1. **Environments** Containing files needed to simulate and review a specific MDP. For each environment at least one MDP, reward function and plot file is present. Current environments are the inverted pendulum, cart-pole, mountain car and the acrobot.
2. **Library** This folder contains all algorithms, basis functions and other solver needed to facilitate learning
3. **Experiments** Folder containing all the initialization files and the data of experiments
4. **Plots** General plotting files for the thesis and the paper.

This folder furthermore contains `start.m` which initializes the toolbox. More info is found in the read-me file.

B-1 Library

The `library` folder contains the folders `Algorithms`, `Approximators` and `LTIToolbox`. The first two are discussed hereafter. The LTIToolbox is programmed by Verhaegen and Verdult (2007) and not discussed. The folder itself contains some general files

B-2 Environments

As mentioned above, different environments were programmed. The inverted pendulum contains the most functions.

Function	Description	Used in Chapter
arrow	Plots arrows	
batchstarer#	Runs Matlab scrips in the folder BATCHTODO#	
circle	Plots circles	
ConvToOld	Converts renamed variables to old names (see ReadMe)	
ConvertVariables	Converts old variable names to new variable names	
cubicfit	Cubic interpolation	
gausspolicy	Gaussian policy	4
gausspolicyLikelyhood	Computes the likelihood that an action is selected for a Gaussian policy	
gibbspolicy	Gibbs policy	
LRateConstAct	Constant learning rate schedule for the actor	all experiments
LRateConstCri	Constant learning rate schedule for the critic	4
movavg	Moving average filter	
ode4_ ti	ODE4 solver	all experiments
parsave	Saves data in parallel loops	4
PlotConfidenceBounds#	A number of files to plot confidence bounds of simulations	all experiments
sat	Saturation function	6
simEpisode	Simulates a trial	4
stoploop	Stops the batchstarter files	
wrap	Wraps angle in $[-\pi, \pi]$	6

Algorithm	Description	Used in Section
aKNAC	averaged Kalman Natural Actor Critic (Pietquin & Geist)	
CurveSearch	The CurveSearch algorithm	not made the thesis
GSEARCH	The GSEARCH algorithm (Baxter & Barlett)	not made the thesis
ImportanceSampingNACLin	Natural Actor-Critic with Experience replay for linear approximators	
mlac	Model Learning Actor Critic	6
NaturalActorCritic	Natural Actor-Critic by Peters et al.	4
NonUpdateVAC	Vanilla Actor Critic without an actor update	5
NonUpdateVACwithHMDP	Vanilla Actor Critic without an actor update with HMDP	5
Qlearning	Approximate Q-learning	
rmac	Reference Model Actor Critic	
SARSA	Approximate Sarsa	6
SARSA_LSTD	Approximate Sarsa with LSTD critic	
SARSA_runPol	Simulate learned policy of approximate Sarsa	
StandardActorCritic	Standard Actor-Critic (SAC)	
actorcriticLSTD	Standard ACcor-Critic using LSTD	
actorcriticLSTDLambda	Standard Actor-Critic using LSTD(λ)	
actorcriticMomentum	Standard Actor-Critic with momentum update	
actorcriticOSA	Standard Actor-Critic with 'Optimal Stepsize Algorithm critic learning rate	
actorcriticXKTD	Standard Actor-Critic with critic estimated with eXtended Kalman TD	
TDNACexperienceReplay	TD-Natural Actor-Critic with experience replay	
TDNaturalActorCritic	TD-version of Natural Actor-Critic	4
TDVanillaActorCritic	TD-version of Vanilla Actor-Critic	4
VACwithBFGS	Vanilla Actor-Critic with BFGS Hessian estimation	5
VanillaActorCritic	Vanilla Actor-Critic	4
VanillaActorCriticMomentum	Vanilla Actor-Critic with momentum actor update	
VGBP_LLR	Value-Gradient Based Policy with LLR critic	6
VGBP_LSTD	Value-Gradient Based Policy with LSTD critic (after each step k)	6
VGBP_LSTD_EpUpdate	Value-Gradient Based Policy with episodic LSTD update	6
VGBP_actionse...	Solution to the optimization problem of Value-Gradient Based Policy	6
VGBP_td	Value-Gradient Based Policy with TD-critic	6
VGBP_tdLin	Value-Gradient Based Policy with TD-critic that is LIP	6
VGBP_SimEpisode	Simulates a learned policy for Value-Gradient Based Policy	6

Approximator (function)	Description	Used in
four	Fourier basis functions (sinusoids)	
fourGen	Function that generates the Fourier basis functions	
IFunctionApprox	Generates handles used by the LLR of Grondman	6
LocLinReg	Local linear regression of Grondman.	6
LUTbf	Lookup table basis function, obtains feature vector from a lookup table	
pbf	Polynomial basis function	
rbf	Radial Basis Function	
rbfDer	Calculates $\nabla_s \phi(s)$ for RBF's	
rbfGen	Generates c and B used in rbf	
rbfDiag	Radial Basis Function with diagonal scaling matrix B and derivative	all experiments
rbfDiagGen	Generates parameters used in rbfDiag	all experiments
rbfLUT	Lookup table version of rbf partitioning	
rbfLUTGen	Creates lookup table from rbf partitioning	
rbfSym#	Symmetrically defined rbfs (yields very, very fast learning)	
returnstates	Returns the state as BFs	
returnstatesDer	Derivative function of returnstates	
tbf	Triangular basis function	
tbf	triangular basis function	
tbfEquispaced	Equispaced verssion of tbf	
tbfGen	Generates properties used in tbf	

B-3 Experiments

All the files to run experiments and all the results are contained in this folder. It has the following subfolders:

- `Acrobot` A single run file for the acrobot. This algorithm still needs to be tuned to work.
- `batchCOMPLETE` Completed files. A variety of runned simulation files. Most functions use parallel loops.
- `batchTODO#` Empty folders used for `batchstarter#`
- `CartPole` A single run file for the cart-pole system. This algorithm still needs to be tuned to work.
- `Goalkeeper` The files to run goalkeeper experiments, containing:
  - Drivers.
  - Initialization files.
  - Real time versions of VGBP for the goalkeeper.
  - plotting files.
  - Simulation and identification data.
- `MountainCar` Examples of SAC and VGBP on the mountain car problem
- `Pendulum` Files for simulations of the inverted pendulum. Each algorithm has its own folder containing the simulation data. Examples of all algorithms are given by the `Run...` files in the `SimpleRun` folder.
- `Pendulum Setup` file that contains all files and data for the DC-motor setup:
  - Drivers.
  - Initialization files.
  - Real time versions of VGBP for the goalkeeper.
  - plotting files.
  - Experimental data

Bibliography

Sander Adam, Lucian Buşoniu, and Robert Babuška. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 42(2):201–212, 2012.

S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276, 1998.

S. Amari and S.C. Douglas. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference onAcoustics, Speech and Signal Processing*, pages 1213–1216 vol.2.

Robert Babuška. *Knowledge-Based Control Systems*. Delft University of Technology, 2010.

D.V. Batalov. Understanding goals in learning by interaction. In *Proceedings of the 2002 International Joint Conference on Neural Networks, 2002*, volume 2, pages 1510–1515.

Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 2001.

S. Becker and Y. LeCun. Improving the convergence of back-propagation learning with second-order methods. In *Proc. of the 1988 Connectionist Models Summer School*, pages 29–37, 1989.

D.P. Bertsekas and J.N. Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, pages 560 –564 vol.1, 1995.

Justin A. Boyan. Technical update: Least-squares temporal difference learning. In *Machine Learning*, pages 233–246, 2002.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Boca Raton, Florida, 2010.

Richard H. Byrd, Gillian M. Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

Pei Chen. Hessian matrix vs. gauss-newton hessian matrix. *SIAM Journal on Numerical Analysis*, 49:1417–1435, 2011.

William Dabney and Andrew G. Barto. Adaptive step-size for online temporal difference learning. 2012.

F.M. Dekking, C. Kraaikamp, H.P. Lopuhaä, and L.E. Meester. *A Modern Introduction to Probability and Statistics: Understanding Why and How*. Springer Texts in Statistics. Springer, 2010.

Fernando Morgado Dias, Ana Antunes, José Vieira, and Alexandre Mota. A sliding window solution for the on-line implementation of the levenberg-marquardt algorithm. *Engineering Applications of Artificial Intelligence*, 19:1–7, 2006.

Kenji Doya. Reinforcement learning in continuous time and space. *Neural Comput.*, 12(1): 219–245, 2000.

Thomas Furmston and David Barber. An approximate newton method for markov decision processes. CORD Conference Proceedings, 2012.

Chris Gaskett, David Wettergreen, and Alexander Zelinsky. Reinforcement learning applied to the control of an autonomous underwater vehicle. In *In Proc. of the Australian Conference on Robotics and Automation*, pages 125–131, 1999.

Matthieu Geist and Olivier Pietquin. Revisiting natural actor-critics with value function approximation. In *Modeling Decisions for Artificial Intelligence*, volume 6408 of *Lecture Notes in Computer Science*, pages 207–218. Springer Berlin / Heidelberg, 2010.

Ivo Grondman, Lucian Buşoniu, Gabriel Lopes, and Robert Babuška. A survey of actor-critic algorithms. 2011a.

Ivo Grondman, Maarten Vaandrager, Lucian Buşoniu, Robert Babuška, and Erik Schuitema. Actor-critic control with reference model learning. *The 18th IFAC World Congress (IFAC-11)*, 2011b.

Ivo Grondman, Lucian Buşoniu, and Robert Babuška. Model learning actor-critic: Effects of algorithm vs. approximator. 2012a.

Ivo Grondman, Maarten Vaandrager, Lucian Buşoniu, Robert Babuška, and Erik Schuitema. Efficient model learning methods for actor-critic control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 42(3):591–602, 2012b.

M.T. Hagan and M.B. Menhaj. Training feedforward networks with the marquardt algorithm. *Neural Networks, IEEE Transactions on*, 5(6):989 –993, 1994.

Sham Kakade. A natural policy gradient. In *NIPS*, pages 1531–1538, 2001a.

Sham Kakade. Optimizing average reward using discounted rewards. In *Computational Learning Theory*, volume 2111, pages 605–615. Springer Berlin / Heidelberg, 2001b.

Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 579–610. Springer Berlin Heidelberg, 2012.

N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, pages 2619 – 2624 Vol.3, 2004.

Vijay R. Konda and John N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42:1143–1166, 2003.

Yann Lecun, Patrice Y. Simard, and Barak Pearlmutter. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. In *Advances in Neural Information Processing Systems*, pages 156–163. Morgan Kaufmann, 1993.

Frank L. Lewis and Draguna Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *Circuits and Systems Magazine*, 09:32–50, 2009.

David Luenberger and Ye. *Linear and Nonlinear Programming*. Springer, third edition, 2008.

Peter Marbach and John N. Tsitsiklis. Simulation-based optimization of markov reward processes. Technical report, 1998.

Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):pp. 431–441, 1963.

Francisco S. Melo, Sean P. Meyn, and M. Isabel Ribeiro. An analysis of reinforcement learning with function approximation, 2008.

Jorge J. Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In G. A. Watson, editor, *Numerical Analysis*, pages 105–116. Springer, Berlin, 1977.

Zhen Ni, Haibo He, D.V. Prokhorov, and Jian Fu. An online actor-critic learning approach with levenberg-marquardt algorithm. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2333–2340, 2011.

Jan Peters, 2002. URL http://www.ias.informatik.tu-darmstadt.de/Research/PolicyGradientToolbox.

Jan Peters and Stefan Schaal. Natural Actor-Critic. *Neurocomputing*, 71(7-9):1180–1190, 2008a. Progress in Modeling, Theory, and Application of Computational Intelligenc - 15th European Symposium on Artificial Neural Networks 2007, 15th European Symposium on Artificial Neural Networks 2007.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008b. Robotics and Neuroscience.

W.B. Powell. *Approximate dynamic programming: solving the curses of dimensionality*. Wiley series in probability and statistics. Wiley-Interscience, 2007.

Ananth Ranganathan. The levenberg-marquardt algorithm, 2004.

Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

S J Russell and P Norvig. *Artificial Intelligence, A Modern Approach - 2nd Edition*, volume 82. Prentice Hall, 2003.

F. Sehnke, A. Graves, C. Osendorfer, and J. Schmidhuber. Multimodal parameter-exploring policy gradients. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pages 113–118, 2010.

Jennie Si, Andrew G. Barto, Warren Buckler Powell, and Don Wunsch. *Handbook of Learning and Approximate Dynamic Programming (IEEE Press Series on Computational Intelligence)*. Wiley-IEEE Press, 2004.

J.C. Spall. Adaptive stochastic approximation by the simultaneous perturbation method. *IEEE Transactions on Automatic Control*, 45(10):1839–1853, 2000.

Richard S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning*, pages 9–44. Kluwer Academic Publishers, 1988.

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ML*, pages 216–224, 1990.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, March 1998.

Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. Technical report, 1999.

Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. Technical report, IEEE Transactions on Automatic Control, 1997.

John N. Tsitsiklis and Benjamin Van Roy. On average versus discounted reward temporal-difference learning. In *Machine Learning*, pages 179–191. MIT, 1999.

Ton van den Boom and Bart de Schutter. *Optimization in Systems and Control*. TU Delft, 2009.

Michel Verhaegen and Vincent Verdult. *Filtering and System Identification: A Least Squares Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2007.

Pawel Wawrzynski. Real-time reinforcement learning by sequential actor-critics and experience replay. *Neural Networks*, 22(10):1484–1497, 2009.

Dietrich Wettschereck, David W. Aha, and Takao Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11:273–314, 1997.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Mauricio Zapateiro and Ningsu Luo. Neural network modeling of a magnetorheological damper. In *Proceedings of the 2007 conference on Artificial Intelligence Research and Development*, pages 351–358, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

Glossary

List of Acronyms

AC	actor-critic
AI	Artificial Intelligence
ASP	Adaptive Simultaneous Perturbation
BF	basis function
BFGS	Broydon-Fletcher-Goldfarb-Shanno
FIM	Fisher Information Matrix
GPI	generalized policy iteration
HMDP	Hessian for Markov Decision Process
LIP	linear in the parameters
LLR	local linear regression
LM	Levenberg Marquardt
LSTD	Least-Squares Temporal Difference
MDP	Markov decision process
MSE	mean squared error
NAC	Natural Actor-Critic
RBF	radial basis function
RL	reinforcement learning
SAC	Standard Actor-Critic
TD	temporal difference

VAC	Vanilla Actor-Critic
VGBP	Value-Gradient Based Policy

List of Symbols

symbols

a	action
b	baseline
c	center of radial basis function
d	steady state distribution
e	eligibility trace
f	function
g	gradient
i	index
j	index
k	time instant
q	small constant
r	reward
s	state
w	advantage function parameter
x	input of a function
y	output of a function
A	advantage function
B	scaling matrix of radial basis function
D	nonsingular scaling matrix of Levenberg Marquardt
E	expectation
F	Fisher Information Matrix
G	Riemannian metric tensor
H	Hessian
I	identity matrix
J	Jacobian
K	general counter
M	memory
N	counter
P	probability
Q	state action value function
V	state value function
Z	step bound
$\mathcal{A}$	action space

$\mathcal{N}$	normal distribution
$\mathcal{P}$	transition function
$\mathcal{R}$	reward function
$\mathcal{S}$	state space
α	learning rate
β	local linear model
γ	discount factor
δ	temporal difference
ϵ	minimum angle between natural gradient
ε	exploration
ζ	sample
η	forgetting factor
κ	damping parameter
λ	eligibility trace decay factor
μ	average
θ	critic parameter vector
ϑ	actor parameter vector
π	policy
ρ	return
ϱ	ratio between actual and predicted reduction of f
σ	standard deviation
ϕ	feature vector
χ	small perturbation constant
ψ	angle
Δ	difference
Ω	set of probability distributions

Superscripts

$*$	optimal
$'$	next
π	under policy π
P	process model
T	matrix transpose

Subscripts

a	actor
c	critic
k	at time instant k
F	Fisher Information Matrix
θ	parameterized with θ
ϑ	under parameter vector ϑ/parameterized with
w	parameterized with w
max	maximum
min	minimum