Machine Control Using Radial Basis Value Functions and Inverse State Projection

Sebastian Buck, Freek Stulp, Michael Beetz, and Thorsten Schmitt
Munich University of Technology, Germany
Department of Computer Science IX
Boltzmannstrasse 3, D-85747 Garching
{buck,stulp,beetzm,schmittt}@in.tum.de

Typical real world machine control tasks have some characteristics which makes them difficult to solve: Their state spaces are high-dimensional and continuous, and it may be impossible to reach a satisfying target state by exploration or human control. To overcome these problems, in this paper, we propose (1) to use radial basis functions for value function approximation in continuous space reinforcement learning and (2) the use of learned inverse projection functions for state space exploration. We apply our approach to path planning in dynamic environments and to an aircraft autolanding simulation, and evaluate its performance.

1 Introduction

Many autonomous machine control skills are too complex and laborious to hand code. Such skills must be acquired by suitable learning algorithms. Reinforcement learning has proven to be such a method. But unfortunately, typical real world machine control tasks have a number of problems: their state spaces are high-dimensional and continuous, and it may be extremely difficult to control the machine towards a desired target state by either hand-coded exploration or human control. Thus we are forced to develop novel solutions that can deal with these problems.

Landing an aircraft is such a problem: The state space is high-dimensional and continuous, therefore neither a simple exploration policy nor human control (except for experienced pilots) will reach the target state. Marginal differences in control may cause significant differences in the resulting state. A state space for the simulation of an aircraft must include distances, angles, and velocities (see fig. 1(a)). Discretizing these values will be a problem, because they are often nonlinear in their temporal behavior. Moreover, determining the neighborhood of a state becomes more difficult. Since the size of the state space grows exponentially with the number of dimensions it is impossible to cover the whole state space by exploration. Furthermore, discretization may lead to extremely discontinuous behavior.

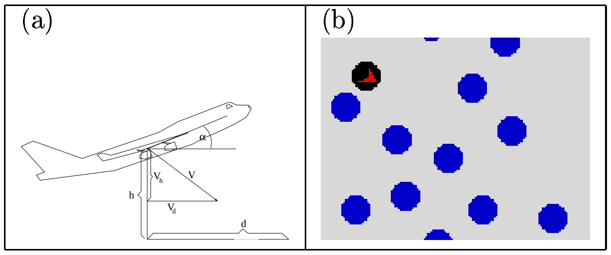

Figure 1: (a): The state of the autolanding task. (b): The path planning problem.

Many machine control problems have been solved successfully by reinforcement learning, as well as tasks with high-dimensional and continuous state spaces. Approximating the value function in reinforcement learning has been surveyed in a number of articles. But how to handle tasks with continuous state spaces where no hand-coded or human-controlled exploration policy ever reaches the target state has received surprisingly little attention so far.

Our contributions in this paper are (1) the use of networks of radial basis functions for the safe approximation of a continuous value function and (2) an inverse neural projection function for the exploration of the state space backwards from the target state. This is an important means for exploration if no trajectories leading to the target state can be found by forward exploration. This is likely in extremely difficult control tasks. To exploit the trajectories given by the backward exploration we build attractors around them. The trajectories are represented by radial basis functions which work as attractors. We then apply gradient descent on the approximated value function.

The remainder of the paper is organized as follows: In sections 2 and 3 we introduce the inverse state projection and the value function approximation using radial basis functions. Section 4 presents results of applying our method to path planning in dynamic environments (fig. 1(b)) and an aircraft autolanding simulation (fig. 1(a)). We discuss related

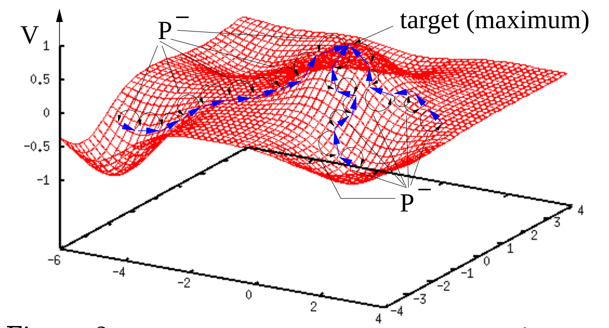

Figure 2: Using the inverse projection $\mathcal{P}^-$ the state space can be explored starting at the target state.

work in section 5. Section 6 closes with a conclusion and future work.

2 Inverse State Projection

Usually the exploration of the state space $\mathcal{S}$ is done by a hand-coded or human-controlled policy. But in extremely difficult control tasks this approach of *forward* exploration is likely to fail because no trajectory to the target can be found. This leads to the idea of a *backward* exploration starting at the target state instead (see figure 2). To safely apply this method we must learn a reliable inverse state projection

$$\mathcal{P}^- : \begin{cases} \mathcal{S} \times \mathcal{A} \to \mathcal{S} \\ \langle \zeta_i, a_{i-1} \rangle \mapsto \zeta_{i-1} \end{cases} \tag{1}$$

that maps a current state ζ_i and its predecessor action a_{i-1} ($\mathcal{A}$ is a set of possible actions) to the predecessor state ζ_{i-1}. $\mathcal{P}^-$ is trained by multi layer neural networks [6, 13] with data acquired by forward exploration.

To generate training data for the learning of the value function $\tilde{\mathcal{V}}$ sequences of backward explorations are made. All states visited during these explorations are given a value corresponding to their temporal distance to the target state or to states from which the value is already known (see figure 3):

$$\mathcal{V}(\zeta) = \begin{cases} +1 \quad \textit{if } \zeta \textit{ is target state} \\ \gamma * \mathcal{V}(\textit{successor}(\zeta)) \quad \textit{else} \end{cases} \tag{2}$$

with γ being a discount factor (e.g. $\gamma = 0.9$). $\tilde{\mathcal{V}}$, telling us how good it is to be in a certain state, is trained with the obtained patterns $\langle \zeta, \mathcal{V}(\zeta) \rangle$ (see section 3). Once $\tilde{\mathcal{V}}$ is computed, it is used to determine the action corresponding to the best-valued successor state to get to the target:

$$a = \arg\max_{a_j \in \mathcal{A}} \tilde{\mathcal{V}}(\mathcal{P}^+(\zeta_i, a_j)) \tag{3}$$

Herein $\mathcal{P}^+$ is a forward projection that maps the current state ζ_i and a possible action a_j to the successor state ζ_{i+1} as follows:

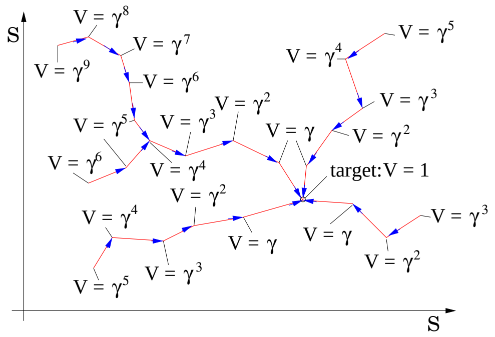

Figure 3: During the exploration of the state space $\mathcal{S}$ the value $\mathcal{V}$ of each state visited is computed according to equation (2). Thus states far from the target get reduced values.

$$\mathcal{P}^+ : \begin{cases} \mathcal{S} \times \mathcal{A} \to \mathcal{S} \\ \langle \zeta_i, a_i \rangle \mapsto \zeta_{i+1} \end{cases} \tag{4}$$

$\mathcal{P}^+$ is trained (with data acquired by forward exploration) by multi layer neural networks [6, 13] as well as $\mathcal{P}^-$ is.

3 Value Function Approximation Using Radial Basis Functions

Using the inverse projection $\mathcal{P}^-$ for the exploration of the state space backwards from the target we get data for learning the value function: A number of trajectories through the state space leading to the target are given. Since we want to learn a value function that leads towards the target state we need to build attractors around these trajectories. Therefore we use networks of radial basis functions [8, 11, 12]. These functions represent a multivariate Gaussian function (RBF-neuron) ϕ_i each. Each RBF-neuron ϕ_i is characterized by the parameters $\vec{\mu_i}$, σ_i, w_i, and θ_i where $\vec{\mu_i}$ is the mean in $\mathcal{S}$, σ_i is the standard deviation, w_i is the height (weight) of the function, and θ_i is the threshold of the neuron:

$$\phi_i(\vec{\zeta}) = w_i * e^{-\frac{|\vec{\zeta} - \vec{\mu_i}|^2}{2\sigma_i^2}} - \theta_i \tag{5}$$

Thus the number of free parameters per RBF-neuron is $d + 3$ where $d = dim(\mathcal{S})$. We approximate $\mathcal{V}$ by a number of such RBF-neurons ($\tilde{\mathcal{V}}$):

$$\tilde{\mathcal{V}} = \sum_{i=1}^{n} \phi_i \tag{6}$$

For the training we set some neurons on the training patterns: $\mu = \zeta$, $w = \mathcal{V}(\zeta)$. We then approximate $\mathcal{V}$ by using the error function

$$E = \frac{1}{2} \sum_{k=1}^{p} (\tilde{\mathcal{V}}(\zeta_k) - \mathcal{V}(\zeta_k))^2 \tag{7}$$

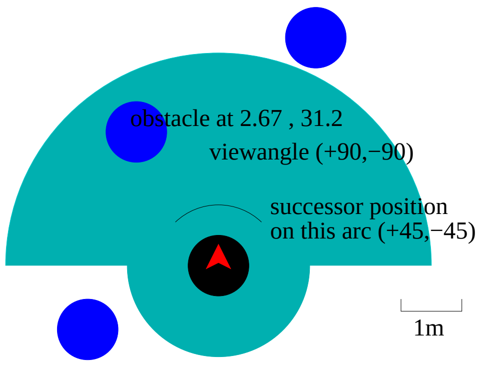

Figure 4: The simulated robot perceives objects up to a distance of 1m all around it and up to 3m in its viewangle.

where p is the number of training patterns. Fortunately $\frac{\delta E}{\delta \vec{\mu}_i}$, $\frac{\delta E}{\delta \sigma_i}$, $\frac{\delta E}{\delta w_i}$, and $\frac{\delta E}{\delta \theta_i}$ can easily be computed (see appendix). Therefore we use gradient descent with adaptive step size computation [13] to learn $\tilde{\mathcal{V}}$.

Using our RBF++ library (C++ library for RBF networks) enables us to make use of some further features such as (1) automatic merging of neighboring RBF-neurons ($|\vec{\mu_i} - \vec{\mu_j}|$ smaller than a certain ϵ), (2) elimination of ineffective RBF-neurons (w_i less than a certain ϵ or $|\vec{\mu_i}|$ bigger than a certain ϵ), (3) automatic insertion of new RBF-neurons (if the error is above a certain ϵ for a certain time) at places with high error etc.

4 Experimental Results

4.1 Path Planning in Dynamic Environments

The path planning problem chosen involves learning to steer a robot (diameter $1m$) through a room of $40*40m^2$ with 250 moving obstacles with a diameter of $1m$ each. The robot *must* move $1m$ per simulation cycle but can choose an angle between -45 and 45 degrees. So actions are chosen from the action space

$$\mathcal{A} = \{1\} \times [-45, +45] \quad (8)$$

The robot gets information on the distance and the angle of obstacles in its viewing angle ($-90 \ldots 90$ degrees) up to a distance of 4 meters. Up to a distance of $1m$ the robot perceives obstacles anywhere around it. Obstacles move at $1m$ per cycle as well. They move to randomly defined targets.

To simplify the problem we define a state by

$$\zeta_o = \langle d, \alpha \rangle \quad (9)$$

with d (α) being the distance to (angle to) obstacle o. $\tilde{\mathcal{V}}$ is then computed by minimizing over all obstacles perceived (O):

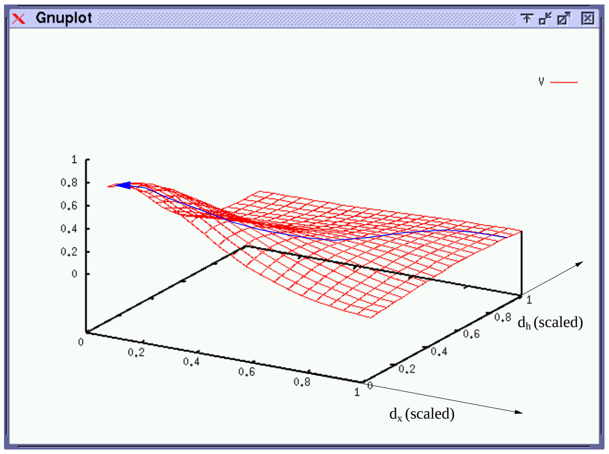

Figure 5: The approximated value function $\tilde{\mathcal{V}}$ represented by radial basis functions. The function is depicted dependent on 2 dimensions only. d_x and d_h are scaled to $[0, 1]$ both.

$$\tilde{\mathcal{V}} = min_{o \in O} \tilde{\mathcal{V}}_1(\zeta_o) \quad (10)$$

where $\tilde{\mathcal{V}}_1$ is the approximated value function for *one* obstacle. When applying $\tilde{\mathcal{V}}$ we heuristically generate 40 possible actions and evaluate all their successor states. That leads to a success rate of 98.8% for static obstacles and to a success rate of 88.6% for dynamic obstacles (around 3000 trials each). Success means that the robot reaches its target without any collision. Regarding the lower success rate for dynamic obstacles one must consider that there are cases where the robot cannot avoid all obstacles by any action. Further exploration was done with static obstacles only. The success rate during exploration was close to zero. Altogether these are encouraging results, especially for the dynamic obstacles. A short animation related to these experiments can be found at *http://www9.in.tum.de/people/buck/pprl1.gif*

4.2 The Aircraft Autolanding Task

While the path planning task described in the previous section does not necessarily need inverse projection the problem introduced here is much more difficult: If an aircraft is landing its velocity both in horizontal and vertical direction must be very small. But to get a small sinking velocity a high horizontal velocity is necessary. Furthermore, the landing position and angle must satisfy some very special constraints. Some airports have only very short landing strips so landing must occur in a limited space. All this makes it inevitable that the policy should plan ahead. Furthermore, the target state can definitely not be reached by any random or simple policy. Even a pilot must have extensive training to be able to land an aircraft safely.

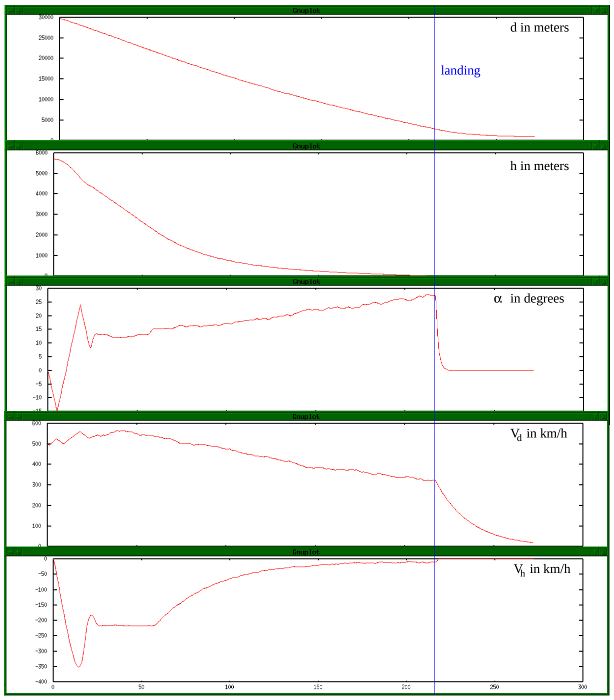

Figure 6: The state of a simulated aircraft over time. The vertical line marks the landing.

In the following simulation we simplify the problem by regarding a five-dimensional state space only. A state

$$\zeta = \langle d_x, d_h, \alpha, V_x, V_h \rangle \tag{11}$$

includes the distance to the end of the landing strip projected on the ground (d_x), the height of the aircraft (d_h), the angle of the aircraft ($\alpha = 0$ means horizontal flight), the horizontal velocity V_x, and the vertical velocity V_h (see figure 1(a)). Even with this simplification it is extremely difficult for a human to safely land an aircraft simulated by the means described in the appendix. In contrast to the work described in [10] our task is not to follow a given trajectory but to *find* a trajectory towards a target state.

The action space is two-dimensional and consists of the power p that controls the propulsive force and the rotational factor r that makes the aircraft rise or sink by adjusting α:

$$a = \langle p, r \rangle \tag{12}$$

p must be in the range of $[0, F_{s_{max}}]$ and $r \in [-10, 10]$, where $F_{s_{max}}$ is the maximal propulsive force of the aircraft (see appendix).

The scenario is the following: an aircraft is placed at ζ_{start}. Then the control policy must land the aircraft at ζ_{target}. Table 1 specifies the constraints for the start and target states.

During forward exploration a simple policy never managed to reach a target state. But using the in-

Variable	unit	range ζ_{start}	range ζ_{target}
d_x	km	$\in [20, 30]$	$\in [0, 3]$
d_h	km	$\in [6, 8]$	$= 0$
α	degrees	$\in [-10, +10]$	$\in [0, +30]$
V_x	km/h	$\in [500, 800]$	$\in [0, 300]$
V_h	km/h	$\in [-200, +100]$	$\in [-25, 0]$

Table 1: Constraints for start and target states.

verse projection $\mathcal{P}^-$ (equation 1) trajectories backwards from the target could be found. From the obtained data a network of radial basis functions is trained. This trained value function $\tilde{\mathcal{V}}$ is partly depicted in figure 5. In more than 500 runs it always attracted the aircraft and lead it towards the target. Figure 6 shows a typical development of the aircrafts' state over time. An animation related to the above experiment can be found at *http://www9.in.tum.de/people/buck/aircraft1.gif.*

The experiments described above show that the proposed approach is able to reliably solve difficult problems in machine control.

5 Related Work

There is a number of applications for reinforcement learning in machine control. However, most of them use variations of Q-learning [18], one of the most popular algorithms in reinforcement learning [15]. Q-learning incrementally learns a state-action value function Q from experience. $Q(\zeta, a)$ computes how good it is to perform action a in state ζ. According to

$$\begin{aligned} Q(\zeta_t, a_t) \leftarrow\ & Q(\zeta_t, a_t) + \\ & \alpha(r_{t+1} + \gamma \max_{a'} Q(\zeta_{t+1}, a') - Q(\zeta_t, a_t)) \end{aligned} \tag{13}$$

Q is updated incrementally where $\gamma \in [0, 1]$ is a decrease factor and r_{t+1} is the reward received for performing action a_t in state ζ_t. This takes place in a discrete state space. To apply Q-learning for tasks in continuous domains like machine control the state space can be discretized [7]. But in general, operating in a discrete state space brings some problems: When using a coarse discretization the control output is not smooth and when using a fine discretization the number of states becomes huge, especially in high dimensional spaces [4].

Nevertheless some successful work on machine control reinforcement learning in discrete state spaces has been done.

In order to both limit the number of states and smooth the control output Moore and Atkeson [9] propose to increase the resolution of the state space in interesting regions while decreasing it in less interesting regions.

A standard approach for reinforcement learning in a continuous state space is to use the gradient of the value function to choose an action [19]. Bertsekas and Tsitsiklis [1] propose to use neural networks [6] for the approximation of the value function. But only replacing the discrete value function by a continuous approximation has been shown to fail [2]. Thrun and Schwartz [17] find the *overestimation phenomenon* to be the main reason for that. Overestimation results from noise which is likely in real world applications and since Q-learning is an incremental algorithm there is the danger of accumulation of such errors. Approaches that overcome these problems include the HEDGER algorithm [14] and instance based reinforcement learning [5].

Another problem reported in the literature is the discontinuity problem [16]: What happens if the optimal value function is non-continuous? Nearly all common approximators will fail here. But in practice this case seldom appears.

6 Conclusions and Future Work

The work described in this paper is a non-incremental algorithm for reinforcement learning in multi-dimensional continuous state spaces. The main contributions are an appropriate method for the safe estimation of the value function, a neural forward projection function and an inverse neural projection function.

In contrast to previous work our method does not suffer from possible overestimation resulting from incremental Q-learning. Our value function successfully reproduces trajectories by Gaussian attractors.

Experiments have shown that our method is a suitable means for difficult machine control tasks. However several improvements remain for the future: One can imagine to regard the standard deviations of the RBF-neurons independently, which makes ellipsoid functions possible. This may lead to a higher accuracy of the approximation and may save some RBF-neurons by the drawback of having more parameters per RBF-neuron. Further exploration could be realized as a combined process of forward and backward exploration. This may lead to more diverse trajectories which is necessary (because of the exponential increase of the state space by the number of dimensions) to more efficiently cover the state space by trajectories.

Appendix

A Simulation Details of the Autolanding Task

The simulator used for the experiments in section 4 relies only on the basic physical rules concerning forces and acceleration. It cannot be compared to a real flight simulator. Nevertheless the behavior of the aircraft, to a great extent, is realistic. The following forces are taken into account (see figure 7(a)):

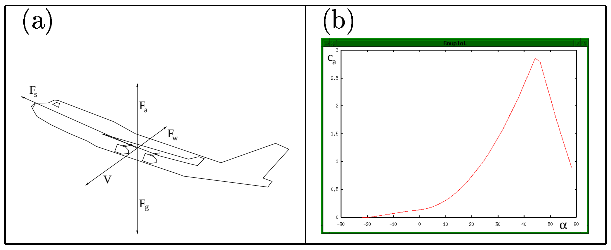

Figure 7: (a): Forces. (b): c_a as a function of α.

Gravitation The force of gravity is computed by $F_g = mg$, where m is the mass of the aircraft and g is the earth constant.

Propulsive Force The propulsive force F_s is directly controlled by the policy ($F_s = p$). It ranges from 0 to $F_{s_{max}}$ (see table below for constants).

Buoyancy Buoyancy is computed by

$$F_a = \frac{1}{2}\rho c_a(\alpha) A_F |\vec{V}|^2 \tag{14}$$

where ρ is the atmospheric pressure, $c_a(\alpha)$ is the buoyancy factor (see figure 7(b)), A_F is the vane area of the aircraft, and V is the velocity of the aircraft.

Air Drag The force of air drag is computed by

$$F_w = -\frac{1}{2}\rho c_w A |\vec{V}|^2 \tag{15}$$

where ρ is the atmospheric pressure, c_w is the air drag coefficient of the aircraft, A is the cross section of the vanes of the aircraft, and V is the velocity of the aircraft.

Acceleration The velocity of the aircraft is computed by

$$\vec{V}_{t+\Delta t} = \vec{V}_t + \frac{(\vec{F}_g + \vec{F}_s + \vec{F}_a + \vec{F}_w) \cdot \Delta t}{m} \tag{16}$$

where Δt is set to one second. Thus the acceleration directly depends on the forces.

Changes in State The change in state depends on the velocities and is computed as follows:

$$\overrightarrow{\langle d, h \rangle}_{t+\Delta t} = \overrightarrow{\langle d, h \rangle}_t + \vec{V}_{t+\Delta t} \cdot \Delta t \tag{17}$$

Once again Δt is set to one second. The angle of the aircraft (α) directly depends on the control surface devices which are controlled by the policy:

$$\alpha_{t+\Delta t} = \alpha_t + r_t \tag{18}$$

Table of Used Constants For our simulation certain constants are used. Each constants' value and meaning is described in the following table. Most values correspond to the values of a filled real Boeing 747.

Var.	Meaning	Value	Unit
ρ	atmospheric pressure	1.3	kg/m^3
c_w	air drag coefficient	2.0	-
c_a	buoyancy constant	$f(\alpha)$	-
g	earth constant	9.81	m/sec^2
A	cross section of vanes	10.0	m^2
A_F	vane area of a 747	528.20	m^2
m	mass of a filled 747	350000	kg
$F_{s_{max}}$	max. propulsive force ($4 \times 257.6\ kN$) (engine GE CF6-80C2B1F)	1030.4	kN

The software of the simulation is accessible at *www9.in.tum.de/people/buck/aircraft_simu.tar.gz* and may be used for research purpose but without any support or warranty. It is written in C++ and runs under Linux, Solaris, and HP-UX.

B Gradient Descent with Radial Basis Functions

To apply the backpropagation algorithm we use the derivations of the error function (equation 7). They are given by

$$\frac{\delta \vec{E}}{\delta \mu_i} = \sum_k (\tilde{\mathcal{V}}(\zeta_k) - \mathcal{V}(\zeta_k)) * \frac{\vec{\zeta_k} - \vec{\mu_i}}{\sigma_i^2} * w_i * e^{-\frac{|\vec{\zeta_k} - \vec{\mu_i}|^2}{2\sigma_i^2}} \tag{19}$$

for the mean of the radial basis function ($\vec{\mu_i}$) and by

$$\frac{\delta E}{\delta \sigma_i} = \sum_k (\tilde{\mathcal{V}}(\zeta_k) - \mathcal{V}(\zeta_k)) * w_i * \frac{|\vec{\zeta_k} - \vec{\mu_i}|^2}{2\sigma_i^3} * e^{-\frac{|\vec{\zeta_k} - \vec{\mu_i}|^2}{2\sigma_i^2}} \tag{20}$$

$$\frac{\delta E}{\delta w_i} = \sum_k (\tilde{\mathcal{V}}(\zeta_k) - \mathcal{V}(\zeta_k)) * e^{-\frac{|\vec{\zeta_k} - \vec{\mu_i}|^2}{2\sigma_i^2}} \tag{21}$$

$$\frac{\delta E}{\delta \theta_i} = -\sum_k (\tilde{\mathcal{V}}(\zeta_k) - \mathcal{V}(\zeta_k)) \tag{22}$$

for the standard deviation (σ_i), the weight of the function (w_i), and the threshold θ_i.

References

[1] D. Bertsekas and J. Tsitsiklis: *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[2] J. Boyan and A. Moore: *Generalization in Reinforcement Learning: Safely approximating the value function.* In Tesauro, G., D. S. Touretzky, and T. K. Leen (eds.), Advances in Neural Information Processing Systems 7 (NIPS). MIT Press, 1995.

[3] S. Buck, M. Beetz, and T. Schmitt: *Approximating the Value Function for Continuous Space Reinforcement Learning in Robot Control.* Submitted to IEEE/RSJ IROS 2002.

[4] K. Doya: *Reinforcement Learning In Continuous Time and Space.* Neural Computation, 12, 219-245, 2000.

[5] J. Forbes and D. Andre: *Real-time reinforcement learning in continuous domains.* AAAI Spring Symposium on Real-Time Autonomous Systems. 2000.

[6] J. Hertz, A. Krogh, R.G. Palmer: *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[7] Z. Kalmar, C. Szepesvari, and A. Lorincz: *Module Based Reinforcement Learning for a Real Robot.* Proceedings of the 6th European Workshop on Learning Robots, Lecture Notes in AI. 1998.

[8] J. Moody and C.J. Darken: *Fast learning in networks of locally-tuned processing units.* Neural Computation, 1, 281-294, 1989.

[9] A. Moore and C. Atkeson: *The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces.* Machine Learning Journal, 21(3):199-233, 1995.

[10] A. Perez-Uribe : *Reinforcement Learning Aircraft Autolander* . Logic Systems Laboratory, Swiss Federal Institute of Technology-Lausanne, http://lslwww.epfl.ch/~aperez/rlaa.html, 1997.

[11] T. Poggio and F. Giorosi: *A theory of networks for approximation and learning.* AI Memo: No 1140, MIT AI Laboratory, 1989.

[12] M.J.D. Powell: *Radial basis functions for multivariate interpolation: A review.* Proc. of the conference on algorithms for the approximation of functions and data, 1985.

[13] M. Riedmiller and H. Braun: *A direct adaptive method for faster backpropagation learning: the Rprop algorithm*, Proceedings of the ICNN, San Francisco, 1993.

[14] W.D. Smart and L.P. Kaelbling: *Practical Reinforcement Learning in Continuous Spaces.* In Proceedings of the Seventeenth International Conference on Machine Learning, pp. 903-910, 2000.

[15] R.S. Sutton and A.G. Barto: *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[16] M. Takeda, T. Nakamura, and T. Ogasawara: *Continuous Valued Q-learning Method Able to Incrementally Refine State Space.* Proceedings of the IEEE International Conference on Intelligent Robots and Systems, 2001.

[17] S. Thrun and A. Schwartz: *Issues in Using Function Approximation for Reinforcement Learning.* In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, editors, Proceedings of the Connectionist Models Summer School, pp. 255-263, Hillsdale, NJ, 1993.

[18] C.J. Watkins and P. Dayan: *Q-learning.* Machine Learning Journal, 8:279-292, 1992.

[19] P. Werbos: *A menu of designs for reinforcement learning over time.* In Neural Networks for Control, W.T. Miller, R.S. Sutton, and P.J. Werbos, editors, pp 67-95, MIT Press, MA, USA, 1990.