Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)

A Recurrent Control Neural Network for Data Efficient Reinforcement Learning

Anton Maximilian Schaefer
Department of Optimisation and Operations Research
University of Ulm, 89069 Ulm, Germany
Email: Schaefer.Anton.ext@siemens.com
Telephone: +49-89-636-45698
Fax: +49-89-636-49767

Steffen Udluft and Hans-Georg Zimmermann
Department of Learning Systems
Information & Communications, Corporate Technology,
Siemens AG, 81739 Munich, Germany
Email: {Steffen.Udluft,
Hans_Georg.Zimmermann}@siemens.com

Abstract— **In this paper we introduce a new model-based approach for a data-efficient modelling and control of reinforcement learning problems in discrete time. Our architecture is based on a recurrent neural network (RNN) with dynamically consistent overshooting, which we extend by an additional control network. The latter has the particular task to learn the optimal policy. This approach has the advantage that by using a neural network we can easily deal with high-dimensions and consequently are able to break Bellman's curse of dimensionality. Further due to the high system-identification quality of RNN our method is highly data-efficient. Because of its properties we refer to our new model as recurrent control neural network (RCNN).**

The network is tested on a standard reinforcement learning problem, namely the cart-pole balancing, where it shows especially in terms of data-efficiency outstanding results.

I. Introduction

Reinforcement learning problems basically consist of an agent and an environment, with which the agent interacts by carrying out different actions. For each interaction the agent gets a reward, which it uses to optimise its policy, i.e., its future actions. In low dimensions reinforcement learning problems are in general solved by table-based methods, where the value of each action-state-combination is stored [1]. For higher-dimension as well as continuous state or action spaces these methods become unfeasible, among other things due to Bellman's curse of dimensionality [2]. Besides that, data efficiency becomes more and more important as in most real-world applications the amount of available data is very limited. In both cases an optimal system identification of the underlying dynamics is essential. For that reason model-based reinforcement learning approaches experienced an increasing interest during the last years.

In this paper we present a new model-based approach to identify and control dynamical systems of reinforcement learning problems in discrete time. Our method is based on a recurrent neural network (RNN) with dynamically consistent overshooting, i.e., which uses its own predictions as future inputs. RNN allow for a data-efficient identification of dynamical systems in form of high-dimensional, nonlinear state space models and are in principle able to approximate any type of open dynamical system [3]. We extend the RNN by an additional control neural network with the particular task to learn the optimal policy, i.e., the optimal mapping from states to actions, of the reinforcement learning problem. Furthermore we adapt its structure to the reinforcement learning environment, i.e., the mapping of Markov decision processes (MDPs), by adding action and reward clusters. Because of its properties we refer to our new network as recurrent control neural network (RCNN).

There have already been a few attempts to combine reinforcement learning with different kinds of recurrent neural networks, e.g. [4], [5], [6]. Schmidhuber's approach [4] is most similar to ours, but still differs potentially in the neural network model and the algorithm used. Bakker [6] showed remarkable results in combining reinforcement learning with long short-term memory (LSTM)-networks [7]. In contrast to these approaches, the recurrent neural network we use, offers an explicit resemblance, in architecture and method, to reinforcement learning and respectively MDPs [8]. Besides that, our RCNN is trained in two subsequent phases. This has the advantage that in the first step the network only focuses on mapping the problem's dynamics whereas in the second step it concentrates on learning the optimal policy based on the identified system. Furthermore our model not only learns from data but also integrates prior knowledge and structure in form of architectural concepts into the modeling. This provides us with the possibility to learn and map efficiently the full environment of a reinforcement learning problem, which is essential for determining the optimal policy.

The paper consists of six parts. We start with a short introduction into the modelling of open dynamical systems by RNN, which we extend by dynamically consistent overshooting (sec. II). Further we illustrate how MDPs can be mapped by RNN (sec. III). Based on this we develop the recurrent control neural network (RCNN) and present its equations and architecture (sec. IV). Subsequent we test the RCNN on an application to the well-known cart-pole problem where it shows especially in terms of data-efficiency outstanding results and outperforms standard reinforcement learning methods by far (sec. V). We conclude with a short summary and an outlook on further research (sec. VI).

1-4244-0706-0/07/$20.00 ©2007 IEEE

II. Modelling of open dynamical systems by RNN with dynamically consistent overshooting

Open dynamical systems in discrete time can be described as a set of equations, consisting of a state transition and an output equation

$$\begin{aligned} s_t^d &= g(s_{t-1}^d, x_t) && \text{state transition} \\ y_{t+1}^d &= h(s_t^d) && \text{output equation} \end{aligned} \tag{1}$$

where g and h are measurable functions, x_t represents the external inputs, s_t^d the inner states and y_t^d the outputs of the system ($t = 1, \ldots, T$ and $T \in \mathbb{N}$) [9], [10].

Any of those open dynamical systems (eq. 1) can be approximated by an RNN of the form [3]:

$$\begin{aligned} s_t &= \tanh(As_{t-1} + Bx_t + \theta) \\ y_{t+1} &= Cs_t \end{aligned} \tag{2}$$

Here, the state transition equation s_t is a nonlinear transformation of the previous state s_{t-1} and the external influences x_t using weight matrices A and B of appropriate dimension and a bias θ, which handles offsets in the input variables x_t. The network's output y_{t+1} is computed from the present state s_t employing matrix C. It is therefore a nonlinear composition applying the transformations A, B, and C. Note, that the state space of the RNN, s_t, in general does not have the same dimension as the one of the open dynamical system s_t^d.

Training the RNN of equation 2 is equivalent to solving a parameter optimisation problem, i.e., minimising the error between the network's output y_t and the observed data y_t^d with respect to an arbitrary error measure, e.g.:

$$\sum_{t=1}^{T} \left(y_t - y_t^d\right)^2 \rightarrow \min_{A,B,C,\theta} \tag{3}$$

It can be solved by finite unfolding in time using shared weight matrices A, B, and C, which share the same memory for storing their weights, i.e., the weight values are the same at each time step τ of the unfolding and for every pattern t [9], [11]. This guarantees that we have the same dynamics in every unfolded time step τ. By using unfolding in time the RNN can be trained with error backpropagation through time (BPTT) [12], which is a shared weights extension of standard backpropagation [13].

In its simplest form RNN unfolded in time only provide a one step prediction of the variable of interest, y_{t+1}. With regard to reinforcement learning this is generally insufficient, because we want to evaluate the system's performance over a certain period of time and therefore need a sequence of forecasts as an output. For this reason we extend the autonomous part of the RNN into the future ($\tau > t$) by so-called dynamically consistent overshooting , i.e., we iterate matrices A and C a finite number of time steps in future direction and use the network's own predictions as future inputs [10], [11]. With overshooting we also increase the system approximation ability of the RNN, as the learning is forced to place more emphasis on modelling the autonomous dynamics of the network, i.e., overshooting supports the extraction of useful information from input vectors, which are more distant to the output [10].

Dynamical consistency solves the problem of the unavailable external information x_τ in the overshooting, respectively future, part ($\tau > t$) of the network. Neglecting these missing influences would be equivalent to the assumption that the environment of the dynamics stays constant, i.e., that the external influences are not significantly changing, when the network is iterated into future direction. Considering external variables with a high impact on the dynamics of interest, this becomes very questionable and might lead to bad generalisation abilities. In RNN with dynamically consistent overshooting we therefore use the network's own predictions as a replacement for the unknown future inputs. In doing so we can, in an elegant manner, integrate assumptions about the future development of the environment x_τ into the modeling. This implies that in an RNN with dynamically consistent overshooting we do not only forecast the variables of interest y_τ^d but all environment data x_τ. As a side effect this allows for an integrated modeling of the dynamics of interest. Note, that due to shared weights for dynamically consistent overshooting no additional parameters are used. The RNN with dynamically consistent overshooting can be described by the following equations[1]:

$$\begin{aligned} s_\tau &= \tanh(As_{\tau-1} + Bx_\tau + \theta) \\ x_{\tau+1} &= Cs_\tau \\ \sum_t \sum_\tau (x_\tau - x_\tau^d)^2 &\rightarrow \min_{A,B,C,\theta} \end{aligned} \tag{4}$$

This can be easily represented in an architectural form (fig. 1), where in the overshooting part ($\tau > t$) of the network the (dashed) connections between the outputs x_τ and the states s_τ provide dynamical consistency. The dotted links indicate that the network can be (finitely) further unfolded in past and future.

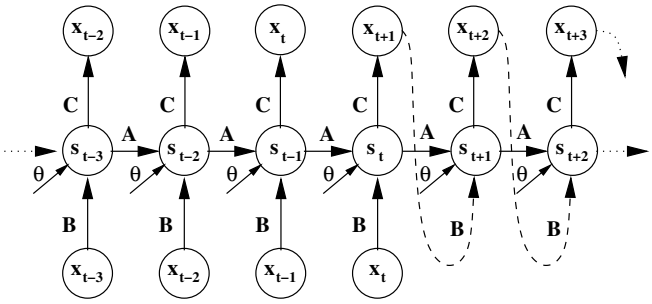

Fig. 1. RNN with dynamically consistent overshooting

Besides its ability to approximate non-linear dynamical systems RNN have the advantage that by using, in contrast to

[1]The parameter τ is hereby always bounded by the length of the (past) unfolding m and the length of the overshooting n, such that we have $\tau \in \{t - m, \ldots, t + n\}$ for all $t \in \{m, \ldots, T - n\}$ with T as the available number of data patterns [11].

e.g. radial basis function (RBF) networks and support vector machines (SVMs), global, sigmoid basis functions, e.g. the hyperbolic tangens, they are well able to cope with higher dimensions and therefore break the curse of dimensionality.

By choosing an appropriate dimension for the internal state s_τ, RNN can also be used to reconstruct the state space of a partially observable dynamical system or to reduce the one of a high-dimensional problem. This has been already successfully applied to solve partially observable Markov decision problems (POMDPs) [8].

III. Mapping MDPs by RNN with dynamically consistent overshooting

A Markov decision process (MDP) can be described by a state space X, an action space A, a (stochastic) transition function $P(x_t, a_t, x_{t+1})$ with $x_t,\ x_{t+1} \in X$ and $a_t \in A$ that models the system's development, and a reward or cost function $c : X \times A \to R$, where R is the space of rewards, the agent receives for choosing action a_t in state x_t. The goal is to determine an optimal policy $\pi^* : X \to A$ that maximises the expected cumulated or average reward function c for each state x_t. In the following we consider deterministic MDPs in discrete time with X and A continuous.

Modelling of an MDP with an RNN follows the idea of mapping the process's dynamics, i.e., the transition function $P(\cdot)$, by a high-dimensional nonlinear system equation. We therefore provide the RNN with the states x_t of the MDP as environmental variables and targets. The actions are given to the network as separate inputs. In doing so we get, analogues to the MDP, the sequence of the (environmental) states x_τ and the subsequent actions a_τ on the input side of the RNN finitely unfolded in time. In the equations of the RNN with dynamically consistent overshooting this is implemented, also with regard to the later extension to the recurrent control neural network (sec. IV), by the inclusion of an additional internal pre-state p_τ:

$$\begin{aligned} s_\tau &= \tanh(\mathbf{I}p_\tau + Da_\tau + \theta) \\ x_{\tau+1} &= Cs_\tau \\ \text{with}\quad p_\tau &= As_{\tau-1} + Bx_\tau \end{aligned} \tag{5}$$

The pre-state p_τ aggregates the information from the previous internal state $s_{\tau-1}$ and the external MDP state x_τ. It has the same dimension as the internal state s_τ and is connected to it by a fixed identity matrix $\mathbf{I}$, which is not learned during training. s_τ has the action a_τ as an input and is used for the calculation of the expected next state of the MDP $x_{\tau+1}$. Here D is an additional matrix of appropriate dimension, which handles the influence of the actions a_τ on the internal state s_τ. The actions a_τ are also given to the RNN as future inputs $(\tau > t)$ because they directly influence the MDP's dynamics but cannot or should not be learned by the network. Figure 2 depicts the resulting RNN architecture.

To ensure a good exploration of the state space X of the MDP, the network should be trained with random actions a_τ.

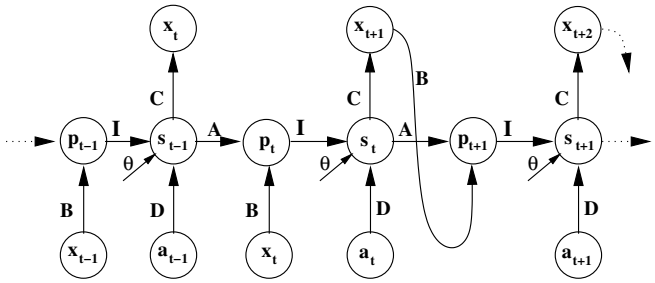

Fig. 2. Mapping an MDP by an RNN with dynamically consistent overshooting

Otherwise the learned dynamics can be dependent on a certain policy π as the system only stays on the related trajectory.

IV. Recurrent Control Neural Network (RCNN)

The Recurrent Control Neural Network (RCNN) is a recurrent neural network, which is able to identify and to control the dynamics of a reinforcement learning or optimal control problem. Its principal architecture is based on the RNN with dynamically consistent overshooting (sec. III), which is extended by an additional control network and an output layer, which incorporates the reward function. Overall its integrated structure follows the idea of mapping the complete reinforcement learning problem, including the underlying MDP, within one network.

The additional and integrated control network has the form of a three layer (input, hidden, output) feedforward neural network. Despite other (more extensive) topologies would be possible, this already allows us to model any arbitrary control function [14]. As we want to predict the optimal action a_τ, the control network is only applied in the present and overshooting part of the RCNN $(\tau \geq t)$ (fig. 3, dashed part). In the past unfolding $(\tau < t)$ the RCNN is provided with the last actions taken.

The control network uses the values of the pre-state p_τ (sec. III), which combines the information of the previous state $s_{\tau-1}$ and the environmental observables x_τ, as inputs. As an output it determines the next action or control variables a_τ. Putting this into equations the control network has the form $(\forall \tau \geq t)$:

$$\begin{aligned} a_\tau &= f(F\tanh(Ep_\tau + b)) \\ \text{with}\quad p_\tau &= As_{\tau-1} + Bx_\tau \end{aligned} \tag{6}$$

where E and F are weight matrices of appropriate dimension, b is a bias and f an arbitrary activation function, which can be used to scale or limit the output and accordingly the action space. The hidden state (fig. 3) of the control network is denoted by $r_\tau (= \tanh(Ep_\tau + b))$.

The RCNN has to fulfill two different tasks, the identification of the problem's dynamics and the optimal control, and is hence trained in two consecutive steps. Note, that this distinguishes our approach from other work on reinforcement

learning and recurrent neural networks, e.g. [6], where one usually tries a combined learning of both tasks in one step. Besides that, the training of our network is offline on the basis of previous observations.

In the first step the RCNN is limited to the identification and modeling of the dynamics of the underlying MDP. It is consequently reduced to the RNN with dynamically consistent overshooting (eq. 5 and fig. 2). Hence, the optimisation task of step one takes on the following form:

$$\begin{aligned} s_\tau &= \tanh(\mathbf{I}p_\tau + Da_\tau + \theta) \\ x_{\tau+1} &= Cs_\tau \\ \text{with} \quad p_\tau &= As_{\tau-1} + Bx_\tau \end{aligned} \tag{7}$$

$$\sum_t \sum_\tau (x_\tau - x_\tau^d)^2 \rightarrow \min_{A,B,C,D,\theta}$$

In the second step all connections coding the dynamics, which have been learned in the first step, in particular matrices A, B, C, and D and the bias θ, get fixed, i.e., their weights are not changed during further training. In return the integrated control network with the matrices E and F and the bias b is activated (fig. 3, dashed connections). These are the only tuneable parameters in this training step. Besides that, as in this step the RCNN's task is to learn the optimal policy, i.e., sequence of actions, it does not get any future actions as external inputs (fig. 3). Furthermore in the past unfolding ($\tau < t$) the output-clusters are taken away, because they were only needed for the identification of the system dynamics in step one. In the present and future part ($\tau \geq t$) of the network the error-function (eq. 7) of the output clusters gets replaced by the reward function. Architecturally this is realised by additional reward clusters R_τ, which are connected to the output clusters by a problem specific, on the reward function $c(\cdot)$ dependent, and fixed matrix G, e.g. equations 9 and 10, as well as a possible activation function h within the output clusters x_τ. In other words the RCNN maps the reward function $c(\cdot)$ of the underlying reinforcement learning problem by coding it in a neural architecture. This implies that R_τ does not only have to be calculated on the basis of the output cluster x_τ but can easily be extend to a more general setting, which usually comes along with a more complicated network architecture.[2]

The weights are now only adapted according to the back-propagated reward from the reward clusters R_τ ($\tau > t$). This follows the idea that in the second step we do not want to identify the problems dynamics but learn a policy, which maximises the reward given the system dynamics modelled in step one (eq. 7). Note that, in doing so the learning algorithm changes from a descriptive to a normative error function.

Summarising, step two can be represented by the following set of equations (eq. 8). Here, the control matrices, E and F and the related bias b, which are learned in this step, are italicised.

$$\begin{aligned} s_\tau &= \tanh(\mathbf{I}p_\tau + Da_\tau + \theta) \\ R_{\tau+1} &= Gh(Cs_\tau), \quad \forall \tau \geq t \\ \text{with} \quad p_\tau &:= As_{\tau-1} + Bx_\tau \\ \text{and} \quad a_\tau &:= f(F\tanh(Ep_\tau + b)) \quad \forall \tau \geq t \end{aligned} \tag{8}$$

$$\sum_t \sum_{\tau \geq t} c(R_\tau) \rightarrow \min_{E,F,b}$$

The architecture of the RCNN in the second step, i.e., during learning of the optimal control, is depicted in figure 3.

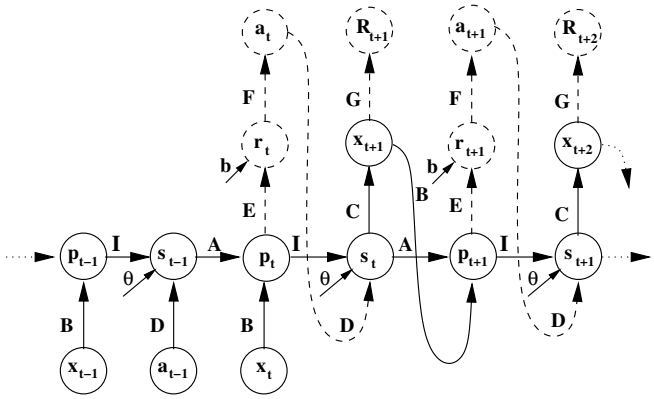

Fig. 3. Learning the optimal policy by a Recurrent Control Neural Network

In both steps (eqs. 7 and 8) the RCNN is trained on the identical set of training patterns T and with BPTT [12]. Concerning the second step this means in a metaphoric sense that by backpropagating the error of the reward function $c(\cdot)$, the algorithm fulfills the task of transferring the reward back to the agent.

The RCNN ideally combines the advantages of an RNN with dynamically consistent overshooting for identifying the problem's dynamics and a three layer control neural network for learning the optimal policy. In doing so, we can benefit from a high approximation accuracy and therefore control complex dynamics in a very data-efficient way. Besides that, we can easily scale into high-dimensions or reconstruct a partial observable environment [8]. Furthermore, out of the construction of the RCNN, it can well handle continuous state and action spaces.

V. A DATA-EFFICIENT SOLUTION TO THE CART-POLE

The cart-pole problem has been extensively studied in control and reinforcement learning theory. For more than 30 years it served as a benchmark for new ideas, as it is easy to understand and also quite representative for related questions. For simplicity we also use it as a first benchmark for our new model.

The classical cart-pole problem consists of a cart, which is able to move on a bounded track and trying to balance a pole

[2] As an alternative it is also possible to learn the reward function explicitly, which is especially of interest in cases, where $c(\cdot)$ is not known or only incompletely specified in the problem setting. This can be realised by a further additional three layer neural network with the output of the RCNN as inputs.

on its top. The system is fully defined through four variables ($t = 1, \ldots, T$):

$$\begin{array}{lcl} \chi_t & := & \text{horizontal cart position} \\ \dot{\chi}_t & := & \text{horizontal velocity of the cart} \\ \alpha_t & := & \text{angle between pole and vertical} \\ \dot{\alpha}_t & := & \text{angular velocity of the pole} \end{array}$$

The problem's system dynamics is given by

$$\begin{bmatrix} M+m & ml\cos\alpha_t \\ ml\cos\alpha_t & \frac{3}{4}ml^2 \end{bmatrix} \begin{bmatrix} \ddot{\chi}_t \\ \ddot{\alpha}_t \end{bmatrix} - \begin{bmatrix} ml\dot{\alpha_t}^2\sin\alpha_t \\ mgl\sin\alpha_t \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}$$

where M and m are the masses of the cart and pole respectively, l is half the length of the pole, g the acceleration due to gravity and F the force applied to the cart.

The goal is to balance the pole for a preferably long sequence of time steps without moving out of the limits. Possible actions are to push the cart left or right with a constant force F. The pole tilts when its angle α_t is larger than 12 degrees. Either then or when the cart hits one of the boundaries the system is punished with a negativ reinforcement signal. In all other cases the reward is zero.

The problem has been completely solved in the past, e.g. [1]. Still all successful methods need quite a large number of training patterns to find a solution. In contrast, for real world applications training data is in general very limited. Consequently methods, which require less training data, i.e., which are more data-efficient, to solve the problem, are preferable. In the following experiment we put a special focus on data-efficiency and show that the RCNN achieves outstanding results. Similar tests have been reported in [15], but with a slightly different dynamics and an extended action space.

A. Model description

We use an RCNN as described in section IV with an unfolding of 10 steps into the past and 30 into the future. This gives the network both, a memory length, which is sufficient to identify the dynamics and an overshooting length, which enables it to predict the consequences of its chosen actions. The internal state dimension, $\dim(s)$, is set to 20 and the hidden state of the control network, $\dim(c)$, to 40 neurons. These dimensions are effectual to generate stable results in terms of system identification and learning the optimal policy. Larger networks in general only require more computational time. Further, the hyperbolic tangens is implemented as an activation function in the clusters a_τ of the integrated control network (eq. 6). This limits the action space of the RCNN to $(-1, 1)$.

For training the RCNN we generated data of different set sizes where the actions were chosen randomly. Here, we varied from the standard setting as we transformed the originally episodic task into a continuous one by keeping the pole or cart at their limits instead of starting a new episode when the pole falls or the cart hits one of the boundaries. This implies the use of a reward function of the form

$$c = \sum_t \sum_{\tau > t} -(g \cdot \chi_\tau + \alpha_\tau)^2 \tag{9}$$

where g is a scaling factor, which balances the error values of the two variables. In our experiment we set $g = 0.1$. According to this, matrix G takes on the form

$$G = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{10}$$

the activation function h in the outputs x_τ is set to identity, and the clusters R_τ get a squared error function with constant targets of 0.

The adaption makes the time series more applicable for the RCNN, in particular the learning with BPTT, but does not simplify the problem; especially as the generated data is only used for training. The learned policy is later tested on the original system and consequently has to cope with the slightly different setting. Here, also the continuous action space of the network gets re-discretised on -1 and 1.

B. Results

We train the RCNN with different amounts of training data. The learned policy is then tested on the original dynamics of the cart-pole where the number of steps N, which it is able to balance the pole, is measured. We use respectively three data sets with $300, 1000, 3000$, 10000, 30000, and 100000 training patterns. For each set we take the median of the performance over 100 different random start initialisations of the cart and pole during testing. The results are given for each set size as the median and average over the values of the respectively three different training data sets (tab. I). We set the maximum number of steps balanced to $max = 100000$, which we consider as sufficient to demonstrate that the policy has solved the problem.

We compare our results to the adaptiv heuristic critic (AHC) algorithm [16], which shows, to our knowledge, very competitive results on the cart-pole problem. As a second benchmark we take (table-based) Q-learning [17], which is one of the commonly used reinforcement learning methods. In contrast to the RCNN we used for both algorithms the standard setting of the cart-pole problem as their application to the modified one (eq. 9) produced inferior results.

# of	RCNN		AHC		Q-Learning	
obs	median	average	median	average	median	average
300	61	100	74	56	61	52
1000	387	33573	124	150	121	121
3000	max	66912	334	312	111	116
10000	max	max	1033	1554	148	163
30000	max	max	2988	9546	193	501
100000	max	max	max	75393	503	624

TABLE I

MEDIAN AND AVERAGE NUMBER OF STEPS THE POLE WAS BALANCED BY THE RCNN, THE AHC, AND THE Q-LEARNING POLICY GIVEN DIFFERENT NUMBERS OF OBSERVATIONS (OBS).

The results (tab. I) clearly indicate that the RCNN can solve the cart-pole problem very data-efficiently. With only 1000 training patterns the average number of steps balanced

is very high. On one of the tested training data sets with 1000 observations even an optimal policy has been learned. With already 10000 observations the maximum number of steps balanced is achieved on the basis of all tested three data sets. In comparison, the AHC needed at least a 100000 observations for finding a satisfying solution. Q-learning required even more observations, as it still failed to balance the pole with the maximum number of observations tested. Consequently by applying the RCNN we can reduce the number of necessary observations by more than 90%. Figures 4 and 5 illustrate the results for the median and average number of steps balanced.

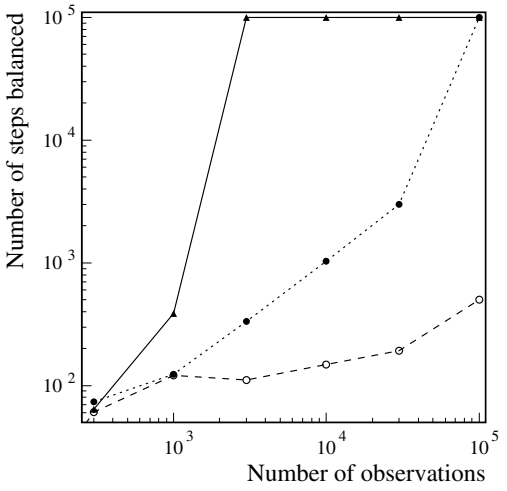

Fig. 4. Numbers of steps balanced with respect to the number of observations taken the median over the different tested data sets. The RCNN (solid line) clearly outperforms the AHC (dotted line) and Q-Learning (dashed line).

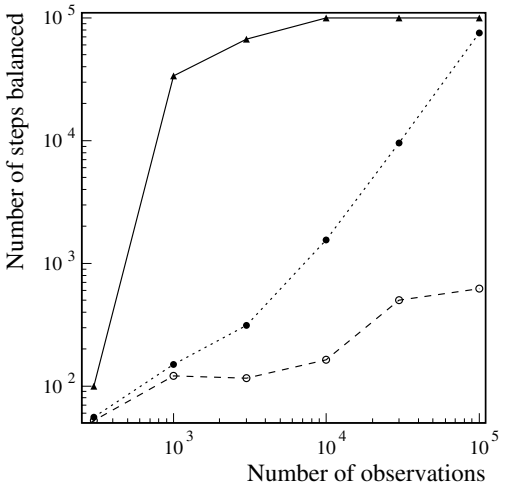

Fig. 5. Number of steps balanced with respect to the number of observations averaged over the different tested data sets. Again the RCNN (solid line) clearly outperforms the AHC (dotted line) and Q-Learning (dashed line).

To examine the stability of the RCNN policy we put a uniform noise on the force F of the action. The task is particularly difficult because the network has not seen any noise during training but its policy has to cope with it during the test on the original dynamics. We take the median and average of the performance over a hundred different random start initialisations of the cart and pole. Table II shows the results for different noise levels on a RCNN policy trained with 10000 observations. It demonstrates that even with a noise level of a 100% the RCNN policy balances the pole in median for the maximum number of steps. Note, that a noise level of more than 100% means that the cart can be pushed into the reverse direction of the one, intended by the policy. This also explains the sharp decrease in performance after increasing the noise to more than 100%. Figure 6 illustrates the robust performance of the RCNN policy.

noise level on F	# of steps balanced	
	median	average
10%	max	max
20%	max	max
30%	max	99953
40%	max	97337
50%	max	98019
60%	max	96937
70%	max	92886
80%	max	88181
90%	max	84191
100%	max	74554
110%	55154	53775
120%	16519	27730
130%	8238	12476
140%	2961	5294
150%	1865	2863
160%	1008	1503
170%	557	939
180%	173	555
190%	76	344
200%	83	242

TABLE II

MEDIAN AND AVERAGE NUMBER OF STEPS BALANCED WITH DIFFERENT NOISE LEVELS ON THE FORCE F BY AN RCNN TRAINED WITH 10000 OBSERVATIONS. EVEN WITH A NOISE OF A 100% THE RCNN IS ABLE TO BALANCE THE POLE THE MAXIMUM NUMBER OF TIME STEPS.

VI. CONCLUSION AND OUTLOOK

In this paper we presented a new model-based reinforcement learning approach. It is based on an RNN with dynamically consistent overshooting, which we briefly introduced. Because of its properties we called it recurrent control neural network (RCNN). Its equations and architecture were described in detail. Here we especially explained the two step learning algorithm. We also pointed out its similarity in structure to MDPs and argued that the combination of an RNN and a three layer neural network is ideal for solving high-dimensional reinforcement learning problems with continuous state and action spaces. On this note the RCNN also offers a good approach to break Bellman's curse of dimensionality [2].

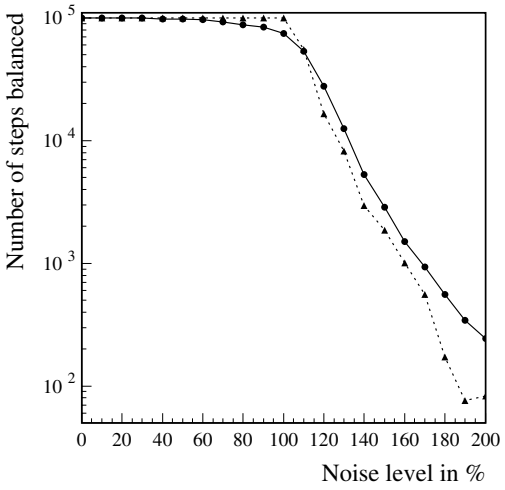

Fig. 6. Median (solid line) and average (dotted line) number of steps balanced with different noise levels on the force F by an RCNN trained with 10000 observations.

The application to the classical cart-pole problem demonstrated the capabilities of the RCNN. Especially in terms of data-efficiency it showed outstanding results and outperformed common reinforcement learning algorithms by far. The policy developed by the RCNN also proved to be very robust, as even with a very high noise level the performance did not deteriorate.

While applying the RCNN to more elaborated industrial and economic problems, further research is done on the network architecture, in particular on a reduction or aggregation of the several different matrices, and an extension of the method to on-line learning. Here we will also refer to higher developed neural network architectures like dynamical consistent neural networks [11]. Besides that we will further analyse and if necessary adapt the RCNN for an application to stochastic problems.

ACKNOWLEDGMENT

Our computations were performed on the Neural Network modeling software SENN (*Simulation Environment for Neural Networks*), which is a product of Siemens AG.

REFERENCES

[1] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. Cambridge, MA: MIT Press, 1998.

[2] R. E. Bellman, *Adaptive Control Processes: A Guided Tour*. Princton, NJ: Princeton University Press, 1961.

[3] A. M. Schaefer and H. G. Zimmermann, "Recurrent neural networks are universal approximators," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN-06)*, Athens, 2006.

[4] J. Schmidhuber, "Reinforcement learning in markovian and non-markovian environments," in *Advances in Neural Information Processing Systems*, D. S. Lippman, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, vol. 3, pp. 500–506.

[5] F. Gomez, "Robust non-linear control through neuroevolution," Ph.D. dissertation, University of Texas, Austin, 2003.

[6] B. Bakker, "The state of mind: Reinforcement learning with recurrent neural networks," Ph.D. dissertation, Leiden University, 2004.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] A. M. Schaefer and S. Udluft, "Solving partially observable reinforcement learning problems with recurrent neural networks," in *Reinforcement Learning in Non-Stationary Environments*, ser. Workshop Proceedings of the European Conference on Machine Learning (ECML-05), 2005.

[9] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New York: Macmillan, 1994.

[10] H. G. Zimmermann and R. Neuneier, "Neural network architectures for the modeling of dynamical systems," in *A Field Guide to Dynamical Recurrent Networks*, J. F. Kolen and S. Kremer, Eds. IEEE Press, 2001, pp. 311–350.

[11] H. G. Zimmermann, R. Grothmann, A. M. Schaefer, and C. Tietz, "Identification and forecasting of large dynamical systems by dynamical consistent neural networks," in *New Directions in Statistical Signal Processing: From Systems to Brain*, S. Haykin, J. Principe, T. Sejnowski, and J. McWhirter, Eds. MIT Press, 2006, pp. 203–242.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in The Microstructure of Cognition*, D. E. Rumelhart and J. L. M. et al., Eds. Cambridge, MA: MIT Press, 1986, vol. 1, pp. 318–362.

[13] P. J. Werbos, *The Roots of Backpropagation. From Ordered Derivatives to Neural Networks and Political Forecasting*. New York: John Wiley & Sons, 1994.

[14] K. Hornik, M. Stinchcombe, and H. White, "Multi-layer feedforward networks are universal approximators," in *Artificial Neural Networks : Approximation and Learning Theory*, H. W. et al., Ed. Cambridge: Blackwell, 1992.

[15] M. Riedmiller, "Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method," in *Machine Learning: ECML 2005*, ser. Lecture Notes in Artificial Intelligence, J. G. et al., Ed., no. 3720. Springer, 2005, pp. 317–328.

[16] R. S. Sutton, A. Barto, and C. Anderson, "Neuron-like adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 13, pp. 834–846, 1983.

[17] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.