SOLVING PARTIALLY OBSERVABLE REINFORCEMENT LEARNING PROBLEMS WITH RECURRENT NEURAL NETWORKS

Anton Maximilian Schäfer and Steffen Udluft

Siemens AG, Corporate Technology,
Information & Communication Division,
Departement Neural Computation
{schaefer.anton.ext,steffen.udluft}@siemens.com

Abstract. In partially observable environments effective reinforcement learning (RL) is still a fairly open question. Most common algorithms fail to produce good results for those problems. However, many real-world applications are characterized by those difficult environments.
In this paper we propose the application of recurrent neural networks (RNN) to identify in a first step the complete state space of the environment out of the only partially available information. In a second step we apply a standard RL algorithm to the reconstructed state space. In doing so we use the capability of RNNs to identify and simulate any dynamical system. Partially observable Markov decision processes (POMDPs) as well as RNNs are both state space models. Out of that reason RNNs are a very appropriate and even obvious method to model reinforcement learning problems. Further we can profit from the wide range of well-established neural network learning and optimization methods.
We show that only minimum information of the reinforcement learning problem is needed to achieve remarkable results. This is due to the fact, that RNN are able to compensate missing information by building up their own internal dynamics and memory. We demonstrate our theoretical results on a variation of the well-known cart-pole problem. For the first time we reduce the observability of this example to one single variable.

1 Introduction

In this paper we present a new approach to identify dynamical systems of partially observable reinforcement learning problems in discrete time. We use recurrent neural networks (RNNs) as they allow for the identification of dynamical systems in form of high dimensional, nonlinear state space models. They offer an explicit modeling of time and memory and are in principle able to model any type of dynamical system [Hay94; MJ99; KK01]. Further the architecture of recurrent neural networks allow for a perfect modelling of the reinforcement learning (RL) environment over a certain number of time steps. Our approach therefore differs from most research directions in a significant but at first sight non-obvious

way. Instead of focusing on algorithms, we put a neural network architecture in the foreground. This distinguishes our networks also from other works on reinforcement learning with recurrent networks [Gom03; Bak04; MCC97]. None of those networks offer this explicit resemblance - in architecture and method - to reinforcement learning. It provides us with the possibility to learn and map efficiently the full environment of the RL problem. There have been some other approaches with similar intentions, e.g. [Sch91], but they all do not have that explicit advantage in architecture and structure.

Besides the architectural advantage we focus on partial observability. We point out that RNNs are highly applicable to those problems as they are able to reconstruct a transformation of the original state space with only a minimum number of available information.

The paper is divided into five parts. Subsequent to the introduction section two gives a short outline about recurrent neural networks. In section three we then demonstrate in detail how we use RNNs for an optimal state space recognition of reinforcement learning problems and the handling of partial observability. Section four underlines our theoretic results by an application to the well known cart-pole problem. A conclusion including an outlook on further research is given in section five.

2 Recurrent Neural Networks (RNN)

Reinforcement learning problems basically consist of an agent interacting with its environment by carrying out different actions. The evolvement of the environment – resulting from the interaction with the agent – can be described as an open dynamical system. For discrete time grids ($t = 1, \ldots, T$ and $T \in \mathbb{N}$) this can be represented as a set of equations (Eq. 1), consisting of a state transition and an output equation [Hay94; KK01]:

$$\begin{aligned} s_{t+1} &= f(s_t, u_t) \quad \text{state transition} \\ y_t \quad &= g(s_t) \qquad \text{output equation} \end{aligned} \tag{1}$$

The state transition is a mapping from the present internal hidden state of the system s_t and the influence of external inputs u_t, to the new state s_{t+1}. In the context of reinforcement learning the inputs u_t are the agent's (partial) information about the environment and his chosen action. The hidden states s_t should not be confused with the current state of the environment. These internal states are rather necessary to develop the evolvement of the environment. The output equation computes the observable output y_t. In our framework the output is equivalent to the resulting change of the system, in other words the subsequent state of the environment.

The task of identifying a dynamic system of Eq. 1 can be stated as the problem to find (parameterized) functions f and g such that a distance measurement (Eq. 2) between the observed data y_t^d and the output y_t of the model is minimal:

$$\sum_{t=1}^{T} \left(y_t - y_t^d\right)^2 \rightarrow \min_{f,g} \tag{2}$$

The identification task of Eq. 1 and 2 can be easily modeled by a *recurrent neural network* of the form

$$\begin{array}{ll} s_{t+1} = \tanh(As_t + c + Bu_t) & \text{state transition} \\ y_t \quad = Cs_t & \text{output equation} \end{array} \tag{3}$$

where A, B and C are weight matrices of appropriate dimensions and c is a bias, which handles offsets in the input variables u_t [ZN01; Hay94].

By specifying the functions f and g as a recurrent neural network with weight matrices A, B and C and a bias vector c, we have transformed the system identification task of Eq. 2 into a parameter optimization problem:

$$\sum_{t=1}^{T} \left(y_t - y_t^d\right)^2 \rightarrow \min_{A,B,C,c} \tag{4}$$

This parameter optimization problem (Eq. 4) is solved by finite unfolding in time using shared weight matrices A, B and C [RHW86; Hay94]. Fig. 1 depicts the resulting spatial neural network architecture [ZNG02]. The recurrent network is trained with error backpropagation through time, which is a shared weights extension of standard backpropagation [RHW86; Hay94].

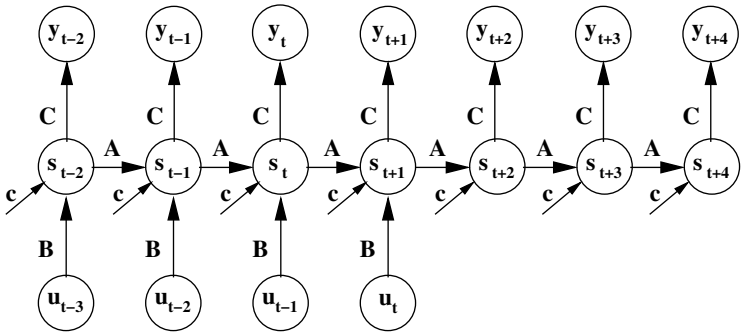

Fig. 1. Recurrent neural network unfolded in time.

The autonomous part of the RNN is extended into the future by so-called overshooting [ZN01], i.e. we iterate matrices A and C in future direction (see Fig. 1). In doing so we get a whole sequence of forecasts as an output. More important overshooting forces the learning to focus on modeling an internal autonomous dynamics of the network, i.e. it supports the extraction of useful information from input vectors which are more distant to the output.[1] In doing so also the learning of false causalities will be decreased. Hence, overshooting regularizes the

[1] Backpropagation learning usually tries to model the relationship between an output and its most recent inputs because the fastest adaptation takes place in the shortest path between input and output.

learning and thus improves the model's performance [ZN01]. Note, that because of shared weights no additional parameters are used.

RNNs are - in contrast to feedforward networks - able to explicitly model memory. This allows for the identification of inter-temporal dependencies. Additionally, as the weights are shared, multiple gradient information is available for the learning. As a consequence, potential overfitting is not as dangerous as for example in the training of feedforward networks. In other words, due to the inclusion of the temporal structure into the network architecture, recurrent networks are applicable to tasks where only a small training set is available [ZN01]. This is especially important for reinforcement learning problems based on a limited number of time steps.

For a more detailed description about RNNs please refer to [ZN01] or [ZGST05].

3 Application of RNNs to RL Problems

As we have argued in section 2 RNNs offer an ideal framework to model dynamical systems. We now use this quality to simulate the system evolvement of a RL problem. Comparing the two methods, RNNs and RL already show a lot of structural resemblance. We want to profit from those to achieve a good system simulation including an optimal state space recognition.

First of all both techniques are state space models. We can therefore easily map the states of a reinforcement learning problem into the architecture of a RNN. The available current state information and the chosen action will be given to the network as external inputs u_t. As targets y_t^d the network gets the available state information of the subsequent time step at a time. The RNN is then able to learn the underlying dynamics of the reinforcement learning problem. Here it is important to note that the training of the RNN is easily done with backpropagation through time [RHW86; Hay94] and therefore - in opposition to an often stated opinion - not a major problem.

A strong advantage of the application of recurrent neural networks to reinforcement learning is the fairly easy handling of partially observable problems. Most RL methods are unable to treat those problems as – with an incomplete information about the environment – they cannot identify the underlying dynamics. In contrast RNNs are still able to learn the dynamics although they only get a small part of the system information. The point is that due to the techniques of unfolding in time and overshooting (see sec. 2) recurrent neural networks can develop autonomously the complete system dynamics. They build up a finite memory and learn intertemporal dependencies out of the available data to compensate the missing information at each time-step [MPKK99]. In doing so they can reconstruct the original state space of the reinforcement learning environment in their internal state s.

In a second step we then apply common reinforcement learning algorithms to that reconstructed internal state space to learn the optimal policy.

4 The partially observable cart-pole

The cart-pole problem has been extensively studied in control and reinforcement learning theory. Since more than 30 years it serves as a benchmark for new ideas, because it easy to understand and also quite representative for related questions. The classical problem has been completely solved in the past. Sutton [SB98] for example demonstrated that the pole can be balanced for an arbitrary number of time steps within a remarkable short training sequence. Consequently the original (simple) problem is not of interest any more.

There are two major directions to make the cart-pole problem more challenging. One is to make the task itself more difficult by taking for example two poles [Gom03] or regarding a two dimensional cart [GM98]. Some of these (increasingly difficult) variations of the problem are summarized in [Wie91]. The other one is to make the original problem only partially observable [MPKK99; Bak02; Gom03]. The latter is the one we will focus on.

4.1 Problem description

The classical cart pole problem consists of a cart which is able to move on a bounded track and trying to balance a pole on its top. This cart-pole system is illustrated in Fig. 2 [MPKK99].

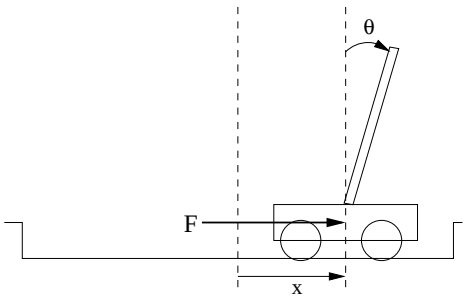

Fig. 2. The cart-pole problem system.

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

$$\begin{array}{ll} x_t := & \text{horizontal cart position} \\ \dot{x}_t := & \text{horizontal velocity of the cart} \\ \theta_t := & \text{angle between pole and vertical} \\ \dot{\theta}_t := & \text{angular velocity of the pole} \end{array} \tag{5}$$

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 θ_t is higher than 12

degrees. Either then or when the cart hits one of the boundaries, the system is punished with a negative reinforcement signal. In all other cases the reward is zero.

As already mentioned the system has been extensively studied in its several forms. Still, so far nobody tried to reduce the observability to only one single variable. When the system was studied as partially observable, one usually omitted the two velocities, $\dot{x}_t$ and $\dot{\theta}_t$, i.e. only the cart's position and the angle between the pole and the vertical where given as inputs [MPKK99; Bak02; Gom03]. Solving this problem is not very difficult because the model or algorithm just needs the memory of one past time step to calculate the missing information.

As we want to fully profit from the advantages of recurrent neural networks (unfolded in time) we only make the horizontal position of the cart, x_t, observable. All other information is absolutely unknown to the system.

In a second experiment we even complicate the problem by adding noise on the only observable variable x_t (Eq. 5), which makes it not only partially observable but also covered by noise. This implies that the learning method cannot absolutely rely on the single information which it receives about the cart-pole's environment but has to extract the true underlying dynamics.

4.2 Model description

To solve the problem described in subsection 4.1 we use a RNN (sec. 2) to develop the full dynamics of the cart pole system. Input u_t and target y_t^d consist of the the horizontal cart position x_t as well as two simple preprocessing transformations of it. The input u_t also contains the agent's action. No other information is observable by the model. We limit the internal state space s_t to four neurons. In doing so we want the network to reconstruct in its internal state space the complete but only partially observable environment (Eq. 5). We unfold the network ten time steps into the past and future. Our results have shown that this memory length is sufficient to identify the dynamics. To make the network independent from the last unfolded time state we use a technique called cleaning noise as a start initialization [ZGST05]. The network is trained by backpropagation through time [RHW86; Hay94] until a minimum error between output and target is achieved.

In a second step we extract the evolved state space from the RNN. We then use a generalized form of Samuel's adaptiv heuristic critic (AHC) algorithm [Sam59], which belongs to the class of temporal difference learning methods [Sut87]. Note, that we have to start the algorithm with an already filled lag structure. Otherwise there is a high probability that the algorithm is faced with an unstable pole in his first learning step as a minimum of ten uncontrolled time steps would be necessary to fill all the lags.

4.3 Results

We used several different test sets to train our networks. As a first result we confirmed that the length of the test set is more important than the number of

training epochs. The more different information about the single input variable the network experiences the more it is able to reconstruct the original (complete) state space.

Fig. 3 points out how well the RNN can identify the underlying dynamics. The four plots show the correlation between the original state space variables of the environment, $x_t, \dot{x}_t, \theta_t, \dot{\theta}_t$, (Eq. 5) and the best linear combination of the reconstructed state space variables $(s_t)_1, \ldots, (s_t)_4$ and their squares $(s_t)_1^2, \ldots, (s_t)_4^2$ in each case. The high correlation for each state space variable demonstrates the reconstruction quality of the RNN. It also strongly supports the use of RNNs for partially observable reinforcement learning problems.

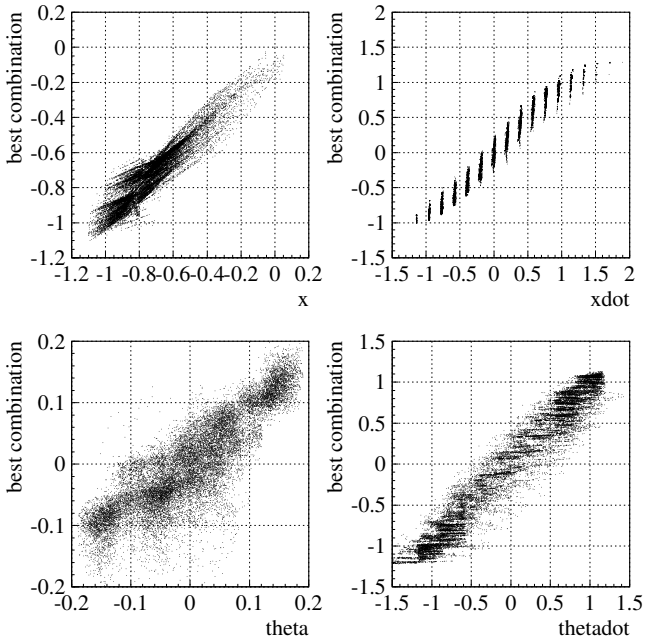

Fig. 3. Correlation between the best quadratic combination of the reconstructed state space variables $(s_t)_1, \ldots, (s_t)_4$ of the RNN and the original ones (Eq. 5).

We compared the results of our approach to a direct application of the AHC algorithm to the problem, which means without using a recurrent neural network in the first step. Note, that no adaptive binning has been used. In both cases we took the discretization of the state space which yielded to the best results.

Fig. 4 plots the achieved number of steps, the pole could be balanced, to the number of trials. We stopped the training as the first method was able to balance the pole for a minimum of 1000 steps. The graphic (Fig. 4) shows how our RNN approach outperforms a direct application of the AHC algorithm.

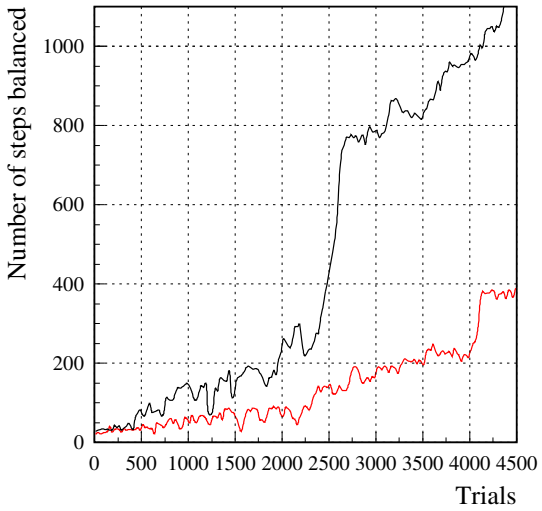

Fig. 4. Comparison of the performance in the partially observable cart-pole problem of our RNN approach (upper curve) to a direct application of the AHC algorithm (lower curve).

The better performance of our approach becomes even more obvious as we add noise to the single observation x_t. We tried different noise levels of $1\%, 2\%$ and 5%. Already for a 1% noise level a direct application of the AHC algorithm fails almost completely to learn the task (Fig. 5 (lower curve)), whereas our RNN approach was – for all tested noise levels – able to balance the pole for at least more than a hundred time steps (Fig. 5 (upper curve)). This result well demonstrates that the RNN is still able to identify and reconstruct the original state space of the environment (Eq. 5) although the only observable information is covered by noise.

In our first tests with noise on the start initialization or on the dynamics itself we achieved similar results. Still these applications need some further research and examination.

5 Conclusion and Outlook

In this paper we focused on solving partially observable reinforcement learning problems with recurrent neural networks. We outlined the principal theory about recurrent neural networks and argued that there is a structural resemblance to reinforcement learning. We further pointed out that with this approach we can profit from the numerous well-established tools and methods of the neural network theory. The application to the well-known cart-pole problem underlined

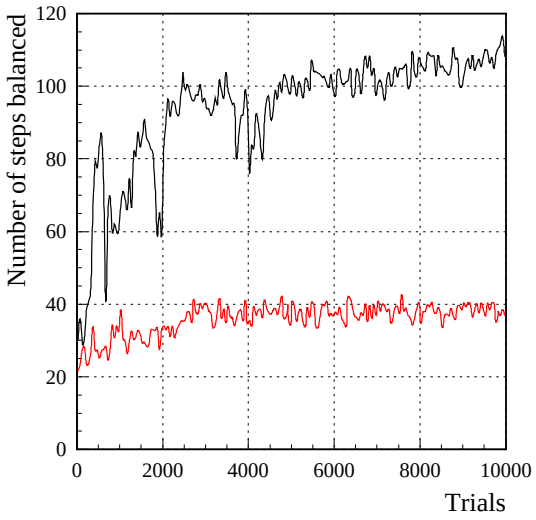

Fig. 5. Comparison of the performance with a 1% noise level on the single observable variable x_t. Our RNN approach (upper curve) is able to balance the pole for at least a hundred time steps whereas a direct application of the AHC algorithm (lower curve) fails almost completely to learn the task. The curve has been averaged over 50 trials.

our theoretical results. We achieved remarkable results and could outperform standard reinforcement learning algorithms.

The presented results and approaches are a first step. We are convinced that the application of recurrent neural networks to reinforcement learning can be improved and extended to other related problems. Therefore we see our results as a starting point to further research. One aspect is the use of higher developed neural network architectures like error correction [ZNG02] or dynamical consistent neural networks [ZGST05]. Another one are high dimensional problems as most reinforcement learning algorithms fail for larger dimensions (Bellmann's "curse of dimensionality"). In contrast recurrent neural networks work well for higher dimensions. They even offer a technique called node pruning to determine an optimal state space recognition which often goes hand in hand with a state space reduction or minimization. First results in this area have shown that the number of state space dimensions can be reduced by at least 50%.

Acknowledgment

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

Bibliography

[And87] Anderson C.: *Strategic Learning with Multilayer Connectionist Representations*, in: Proceedings of the Fourth International Workshop on Machine Learning, Irvine, CA, pp. 103-114, 1987.

[Bak02] Bakker B.: *Reinforcement Learning with Long Short-Term Memory*, in: Dietterich T. G., Becker S. and Ghahramani Z. [Eds.]: *Advances in Neural Information Processing Systems, 14*, Cambridge, MA: MIT Press, pp. 1475-1482, 2002.

[Bak04] Bakker B.: *The State of Mind: Reinforcement Learning with Reucrrent Neural Networks*, Phd Thesis, Leiden University, 2004.

[BT96] Bertsekas D.P. and Tsitsiklis J.N.: *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.

[GM98] Gomez F. and Miikkulainen R.: *2-D Balancing with Recurrent Evolutionary Networks*, in: Proceedings of the International Conference on Artificial Neural Networks (ICANN-98, Skovde, Sweden), Springer, New York, pp. 425-430, 1998.

[Gom03] Gomez F.: *PhD Thesis: Robust Non-Linear Control through Neuroevolution*, Departement of Computer Sciences Technical Report AI-TR-03-3003, 2003.

[Hay94] Haykin S.: *Neural Networks. A Comprehensive Foundation*, Macmillan College Publishing, New York, 1994. second edition 1998.

[KK01] Kolen J. F. and Kremer St. [Eds.]: *A Field Guide to Dynamical Recurrent Networks*, IEEE Press, 2001.

[MJ99] Medsker L. R. and Jain L. C.: *Recurrent Neural Networks: Design and Application*, CRC Press international series on comp. intelligence, No. I, 1999.

[MCC97] Micher O., Clergue M. and Collard Ph.: *Artificial Neurogenesis: Aplications to the cart-pole problem and to an autonomous mobile robot*, in: International Journal on Artificial Intelligence Tools, Vol. 6, No. 4, pp. 613-634, 1997.

[MPKK99] Meuleau N., Peshkin L., Kim Kee-Eung and Kaebling L.P.: *Learning Finite-State Controllers for Partially Observable Environments*, in: Proceedings of the Fifteenth International Conference on Uncertainty in Artificial Intelligence (UAI-99), Morgan Kaufmann, San Francisco, CA, pp. 427-436, 1999.

[NZ98] Neuneier R. and Zimmermann H. G.: *How to Train Neural Networks*, in: *Neural Networks: Tricks of the Trade*, Springer, Berlin, pp. 373-423, 1998.

[RHW86] Rumelhart D. E., Hinton G. E. and Williams R. J.: *Learning Internal Representations by Error Propagation*, in: D.E. Rumelhart, J. L. McClelland, et al., Parallel Distributed Processing: Explorations in The Microstructure of Cognition, Vol. 1: Foundations, Cambridge: MIT Press, pp. 318-362, 1986.

[Sam59] Samuel A. L.:*Some studies in machine learning using the gamecheckers*, in: IBM Journal on Research and Developement, 3, pp. 210-229, 1959.

[Sch91] Schmidhuber J.:*Reinforcement Learning in Markovian and Non-Markovian Environments*, in: Lippman D. S., Moody J.E. and Touretzky D. S.[Eds.]: *Advances in Neural Information Processing Systems, 3*, San Mateo, CA: Morgan Kaufmann, pp. 500-506, 1991.

[Sut87] Sutton R.S.: *Learning to predict by the methods of temporal differences*, GTE Laboratories Incorporated, Technical Report TR87-509.1, Waltham,MA, 1987.

[SB98] Sutton R.S. und Barto A.G.: *Reinforcement Learning, An Introduction (Adaptive Computation and Machine Learning)*, The MIT Press, Cambridge, Massachusetts, 1998.

[Wie91] Wieland A.: *Evolving neural network controllers for unstable systems*, in: Proceedings of the International Joint Conference on Neural Networks (Seatlle, WA), volume II, IEEE, Piscataway, NJ, pp. 667-673, 1991.

[ZN01] Zimmermann H. G. and Neuneier R.: *Neural Network Architectures for the Modeling of Dynamical Systems*, in: A Field Guide to Dynamical Recurrent Networks, Eds. Kolen, J.F.; Kremer, St.; IEEE Press, pp. 311-350, 2001.

[ZNG02] Zimmermann H. G., Neuneier R. and Grothmann R.: *Modeling of Dynamical Systems by Error Correction Neural Networks*, in: Modeling and Forecasting Financial Data, Techniques of Nonlinear Dynamics, Eds. Soofi, A. and Cao, L., Kluwer Academic Publishers, pp. 237-263, 2002.

[ZGST05] Zimmermann H. G., Grothmann R., Schäfer A.M. and Tietz Ch.: *Identification and Forecasting of Large Dynamical Systems by Dynamical Consistent Neural Networks*, in: Haykin S., Principe J., Sejnowski T. and McWhirter J.[Eds.]: *New Directions in Statistical Signal Processing: From Systems to Brain*, MIT Press, forthcoming 2005