ESANN'2007 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2007, d-side publi., ISBN 2-930307-07-2.

The Recurrent Control Neural Network

Anton Maximilian Schaefer[1,2], Steffen Udluft[1], and Hans-Georg Zimmermann[1]

1- Siemens AG, Corporate Technology, Department Learning Systems,
Otto-Hahn-Ring 6, D-81739 Munich, Germany

2- University of Osnabrueck, Neuroinformatics Group,
Albrechtstraße 28, D-49069 Osnabrueck, Germany

Abstract. This paper presents our Recurrent Control Neural Network (RCNN), which is a model-based approach for a data-efficient modelling and control of reinforcement learning problems in discrete time. Its architecture is based on a recurrent neural network (RNN), which is extended by an additional control network. The latter has the particular task to learn the optimal policy. This method has the advantage that by using neural networks we can easily deal with high-dimensions or continuous state and action spaces. Furthermore we can profit from their high system-identification and approximation quality. We show that our RCNN is able to learn a potentially optimal policy by testing it on two different settings of the mountain car problem.

1 Introduction

Reinforcement learning (RL) 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 is used to optimise its policy, i.e., its future actions based on the respective states. Low dimensional and discrete RL problems are generally solved by table-based methods, where the value of each state-action-combination is stored [1]. For continuous state or action spaces these methods become unfeasible. In those cases an optimal system identification of the underlying dynamics and a good generalisation are of avail.

In this paper we present our Recurrent Control Neural Network (RCNN), which is a new model-based approach. Its essential part consists of a recurrent neural network (RNN) with dynamically consistent overshooting [2]. We extend it by an additional control neural network with the particular task to learn the optimal policy of the RL problem. Furthermore we adapt its structure to the RL environment by adding action and reward clusters.

There have already been a few attempts to combine RL with different kinds of recurrent neural networks, e.g. [3, 4]. Schmidhuber's approach [3] is most similar to ours, but still differs substantially in the neural network model and the algorithm used. Besides that, none of the existent approaches offers, in architecture, equations, and algorithms, the same flexibility and a comparable explicit resemblance to RL [5].

We test the RCNN on two different settings of the well-known mountain-car problem [1]. Near-optimal solutions have been reported in [6] and [7]. We show that our RCNN outperforms those and is even as good as a manually tuned potentially optimal one.

2 Recurrent Control Neural Network

The Recurrent Control Neural Network (RCNN) is a recurrent neural network, which is able to identify and to control the dynamics of a RL or optimal control problem. Its principal architecture is based on an RNN, 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 RL problem within one network.

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 RL and recurrent neural networks, e.g. [4], where one usually tries a combined learning of both tasks in one step. It 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. For both steps the training is done offline on the basis of previous observations. Its complete architecture is depicted in figure 1.

In the first step the RCNN is limited to the identification and modeling of the system dynamics and is consequently reduced to an RNN with dynamically consistent overshooting [2] (fig. 1, bold and dotted connections). Hence, analogue to those RNN, the optimisation task of step one takes on the following form:

$$\begin{aligned} s_\tau &= \tanh(\mathbf{I}_J p_\tau + D a_\tau^d + \theta) \\ x_{\tau+1} &= C s_\tau \\ \text{with} \quad p_\tau &= \begin{cases} A s_{\tau-1} + B x_\tau^d & \forall \tau \leq t \\ A s_{\tau-1} + B x_\tau & \forall \tau > t \end{cases} \end{aligned} \tag{1}$$

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

Here, the RCNN's internal state transition equation is a nonlinear transformation of the previous internal state $s_{\tau-1} \in \mathbb{R}^J$, either the environmental state variables $x_\tau^d \in \mathbb{R}^I$ or, due to dynamical consistency, the model's predictions $x_\tau \in \mathbb{R}^I$ for those, and the applied actions $a_\tau^d \in \mathbb{R}^N$ using weight matrices $A \in \mathbb{R}^{J \times J}$, $B \in \mathbb{R}^{J \times I}$ and $D \in \mathbb{R}^{J \times N}$ and a bias $\theta \in \mathbb{R}^J$, which handles offsets in the input variables, where I, J, and N denote respectively the number of environmental variables, hidden neurons and available actions. Note, that at this the Hyperbolic Tangens is applied compenent-wise and $\mathbf{I}_J$ stands for a $J \times J$ identity matrix. In addition to standard RNN [2] out of architectural reasons a pre-state $p_\tau \in \mathbb{R}^J$ with the same time index and dimension as s_τ is inserted, which serves in the second step as an input for the control network. The network output $x_{\tau+1} \in \mathbb{R}^I$ is computed from the internal state $s_\tau \in \mathbb{R}^J$ employing matrix $C \in \mathbb{R}^{I \times J}$. Note, that in this step the actions of the observed training data $a_\tau^d \in \mathbb{R}^N$ are also given to the RCNN as present and future inputs ($\tau \geq t$) because they directly influence the system dynamics but cannot or should not be learned by the network (fig. 1, dotted connections).

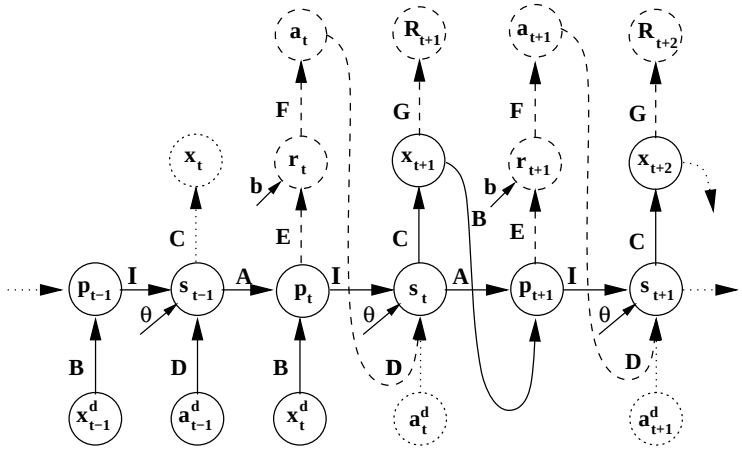

Fig. 1: Learning the optimal policy by the Recurrent Control Neural Network (RCNN). Dotted connections are just used in the first step whereas the dashed parts, the control networks, are only applied in the second one. In phase 1 matrices A, B, C, and D, coding the dynamics are trained. In phase 2 the optimal policy coded in matrices E and F is learned.

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, which has the form of a three layer (input, hidden, output) neural network is activated (fig. 1, dashed connections). It uses the values of the pre-state p_τ, which combines the information of the previous state $s_{\tau-1}$ and the environmental observables x^d_τ, respectively its own predictions 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

$$a_\tau \quad = \quad f(F \tanh(Ep_\tau + b)) \quad \forall \tau \geq t \tag{2}$$

where $E \in \mathbb{R}^{K \times J}$ and $F \in \mathbb{R}^{N \times K}$, with K denoting the number of hidden neurons of the control network, are weight matrices, $b \in \mathbb{R}^K$ is a bias and f an arbitrary, component-wise applied activation function, which can be used to scale or limit the network's action space. The hidden state (fig. 1) of the control network is denoted by $r_\tau \in \mathbb{R}^K$.

As we want to determine the optimal action a_τ, the control network (eq. 2) is applied in the present and overshooting part of the RCNN ($\tau \geq t$), where in this step it does not get anymore future actions as external inputs (fig. 1, dotted connections). In the past unfolding ($\tau < t$) the RCNN is, as in step one, still provided with the actions a^d_τ of the observed training data. Furthermore in the present and past unfolding ($\tau \leq t$) the output-clusters are taken away, because they are only needed for the identification of the system dynamics. In the future

part ($\tau > t$) of the network the error-function (eq. 1) of the output clusters gets replaced by the reward, or respectively cost, function $c(\cdot)$. Architecturally this is realised by additional reward clusters $R_\tau \in \mathbb{R}^L$, where L is the number of variables influencing the reward, which are connected to the output clusters by a problem specific and fixed matrix $G \in \mathbb{R}^{L \times I}$ as well as a possible activation function h within the output clusters x_τ. As an alternative it is also possible to code $c(\cdot)$ directly into an appropriate neural architecture [8].

The weights of the control network are only adapted according to the back-propagated error from the reward clusters R_τ ($\tau > t$). This follows the idea that in the second step we want to learn a policy, which maximises the reward, respectively minimises the costs, given the system dynamics modelled in step one (eq. 1). 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:

$$\begin{aligned}
& s_\tau && = && \begin{cases} \tanh(\mathbf{I}_J p_\tau + D a_\tau^d + \theta) & \forall \tau < t \\ \tanh(\mathbf{I}_J p_\tau + D a_\tau + \theta) & \forall \tau \geq t \end{cases} \\
& R_{\tau+1} && = && Gh(Cs_\tau), \quad \forall \tau \geq t \\
\text{with} \quad & a_\tau && = && f(F \tanh(Ep_\tau + b)) \quad \forall \tau \geq t \\
\text{and} \quad & p_\tau && = && \begin{cases} As_{\tau-1} + Bx_\tau^d & \forall \tau \leq t \\ As_{\tau-1} + Bx_\tau & \forall \tau > t \end{cases} \\
& \sum_t \sum_{\tau > t} c(R_\tau) \rightarrow \max_{E,F,b} &&&&
\end{aligned} \tag{3}$$

In both steps (eqs. 1 and 3) the RCNN is trained on the identical set of training patterns T and with backpropagation through time [9]. 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 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, we can easily scale into high-dimensions or reconstruct a partially observable environment [5]. Furthermore, out of the construction of the RCNN, it can well handle continuous state and action spaces.

3 The mountain car problem

The mountain car problem is fully described in [1]. Its objective is to reach the top of a valley with an underpowered car by gaining kinetic energy through driving forward and backward. The car's continuous state consists of a one dimensional position p and a velocity v. There are three possible actions: wait, full power forward and backward. The reward is one when the car has reached

the top of the hill and zero otherwise. A trivial near-optimal policy, as reported in [6], is to drive the car always full power in the direction of the car's current velocity v.

We used two different settings of the outlined problem, the standard [1] and a slightly simplified one. In the latter we used so-called meta-actions or options [10], which let the agent only take a decision in every fifth time step. This allows the car to travel longer distances in between taking actions. Consequently the car can reach the top of the hill with less decision steps than in the standard setting. For each setting we allowed 100.000 observations for training and tested the learned policy afterwards on the respective simulated dynamics. Note, that for the latter the continuous actions determined by the RCNN had to be re-discretised, which can be seen as an additional difficulty. In the simplified setting the training set for the RCNN was created with random actions. For the standard version we pre-applied ε-greedy prioritised sweeping [1] (with $\varepsilon = 0.5$) to obtain a representative training set because random actions never reached the top of the hill. The applied RCNN was ten time steps unfolded in the past. For the standard setting the future unfolding counted 300 and for the simplified one 50 time steps, which allowed the networks to see at least one goal state within its (finite) future horizon.

We compared our results to the described trivial near-optimal policy [6], standard prioritised sweeping (PS) [1] and the minimum number of decision steps determined by manual tuning. Table 1 summarises our results for the two settings. It shows that the RCNN is able to learn a potentially optimal policy as it is even as good as the manually tuned one. It also outperforms the best results on the problem reported in [7].

	RCNN	PS	Trivial Near-Optimal	Potentially Optimal
Standard	104	144	125	104
Simplified	21	27	26	21

Table 1: Number of decision steps needed by the RCNN, the PS, the trivial near-optimal [6], and the manually tuned potentially optimal policy to drive the car up the hill.

In another experiment we tested the RCNN with regard to data-efficiency and noise robustness on two different settings, one with continuous actions [11], of the well-known cart-pole problem [1], where it also showed outstanding results and outperformed standard benchmarks [8].

4 Conclusion and Outlook

In this paper we presented our Recurrent Control Neural Network, which is a model-based reinforcement learning approach. We argued that the combination of an RNN and a three layer neural network within the RCNN is ideal for solving high-dimensional RL problems with continuous state and action spaces in a data-efficient manner. The application to the mountain car problem demonstrated

the capabilities of the RCNN as it was able to outperform standard methods as well as the cited near-optimal policies. It was even as good as the manually tuned one, which underlines that the RCNN is indeed able to learn a potentially optimal policy.

Further research is done on the network architecture, in particular on a reduction or aggregation of the several different matrices. Besides that we apply the RCNN to more elaborated industrial and economic 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)*. MIT Press, Cambridge, MA, 1998.

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

[3] J. Schmidhuber. Reinforcement learning in markovian and non-markovian environments. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 500–506. Morgan Kaufmann, San Mateo, CA, 1991.

[4] B. Bakker. *The State of Mind: Reinforcement Learning with Recurrent Neural Networks*. PhD thesis, Leiden University, 2004.

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

[6] M. J. A. Strens and A. W. Moore. Direct policy search using paired statistical tests. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001)*, Williams College, MA, 2001.

[7] M. Abramson, P. Pachowicz, and H. Wechsler. Competitive reinforcement learning in continuous control tasks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Portland, OR, 2003.

[8] A. M. Schaefer, S. Udluft, and H. G. Zimmermann. A recurrent control neural network for data efficient reinforcement learning. In *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-2007)*, Honolulu, HI, 2007.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland et al., editors, *Parallel Distributed Processing: Explorations in The Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, 1986.

[10] D. Precup, R. Sutton, and S. Singh. Theoretical results on reinforcement learning with temporally abstract behaviors. In *Proceedings of the 10th European Conference on Machine Learning (ECML)*, pages 382–393, 1998.

[11] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, pages 1107–1149, 2003.