A Neural Reinforcement Learning Approach to Gas Turbine Control

Anton Maximilian Schaefer, Daniel Schneegass, Volkmar Sterzing, and Steffen Udluft
Department of Learning Systems, Information & Communications, Corporate Technology
Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany
Email: {Schaefer.Anton.ext, Daniel.Schneegass.ext, Volkmar.Sterzing, Steffen.Udluft}@siemens.com
Telephone: +49-89-636-44441, Fax: +49-89-636-49767

***Abstract*— In this paper we present a new Neural Network based approach to control a gas turbine for stable operation on high load. We use a combination of Recurrent Neural Networks (RNN) and Reinforcement Learning (RL). We start by applying an RNN to identify the minimal state space of a gas turbine's dynamics. Based on this we determine the optimal control policy by standard RL methods. We proceed to a so called Recurrent Control Neural Network (RCNN), which combines these two steps into one integrated Neural Network.**

Our approach has the advantage that by using Neural Networks we can easily deal with the high dimensions of a gas turbine and due to the high system-identification quality of RNN cope with the, in general, only limited amount of available data. We demonstrate the proposed methods on an exemplary gas turbine model where, compared to standard controllers, it strongly improves the performance.

I. Introduction

Presently and in the foreseeable future alternative and renewable energy sources become more and more important. As a consequence gas turbines are, due to their good ecological properties in comparison with coal-fired power plants, increasingly deployed. Because of their high operational flexibility, they also serve as a compensation for the less predictable energy production of alternative energy sources. To guarantee a smooth operation at high or dynamic loads it is desirable to use sophisticated controllers, which are able to take short and long-term influences into account. For this reason we applied a combination of Recurrent Neural Networks (RNN) [1], [2] and Reinforcement Learning (RL) [3] as a meta-controller, supporting the traditional controller, which was based on physical knowledge and expert's experience. In doing so we achieve a significant improvement in the turbine operation.

Reinforcement Learning [3] is an ideal approach to solve optimal control problems by learning a policy, which maximises or respectively minimises a desired outcome. Relating to the gas turbine, it basically considers a controller (agent) and the turbine's parameters (environment), with which the controller interacts by carrying out different actions. For each interaction it can see the outcome of his action. In other words it gets a reward, which is used to optimise its policy, i.e., its future actions based on the respective states. Low dimensional problems or ones with finite discrete state space, are generally solved by table-based RL methods such as Q-Learning [4], where the value of each state-action-combination is stored, or Temporal-Difference methods on local basis functions [3]. For higher dimensional state or continuous action spaces, like we are confronted with at a gas turbine, these methods become unfeasible. In those cases an optimal system identification of the underlying dynamics is of avail. Besides that, as the amount of available data is generally limited, data-efficiency becomes an important requirement. For that reason model-based RL approaches experienced an increasing interest during the last years.

For controlling a gas turbine we present two new model-based methods based on Recurrent Neural Networks (RNN). In a first approach we apply an RNN to condense the high dimensional state space of the gas turbine, i.e., we identify the minimal dimension of the turbine's dynamics in an information conserving and dynamical way. The reduced state space can then be used by standard (table-based) RL methods. Further we introduce our Recurrent Control Neural Network (RCNN) [5], which basically consists of two parts. An RNN with dynamically consistent overshooting, i.e., which uses its own predictions as future inputs [2], for identifying the dynamics and 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 RL problem. Furthermore, its structure is adapted to the RL environment by adding action and reward clusters.

There have already been a few attempts to control a gas turbine with (Feedforward) Neural Networks [6], [7], [8], but none of those offers the same capabilities and flexibility we gain by combining RL and RNN.

The paper consists of five parts. We start with a short introduction into the modelling of open dynamical systems by RNN, which we extend by dynamically consistent overshooting. We then show how these RNN can be used for an efficient reduction of a high dimensional state space, which can serve as a basis for standard RL methods (sec. II). Subsequently we present the Recurrent Control Neural Network (RCNN), which combines the system dynamics identification and the learning of the optimal control in one integrated network (sec. III). Finally we apply our combined approach and the RCNN on a simulation of a gas turbine and compare their performance (sec. IV). We conclude with a short summary and an outlook on future research (sec. V).

II. State Space Identification by RNN

Open dynamical systems in discrete time can be described by 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 $T \in \mathbb{N}$) [1], [9].

Generally in technical or economic applications the inner state s_t^d of a system is at least partially unknown, i.e., one can only observe a certain number of variables, which might have an influence on the system's dynamics. In contrary, for solving an optimal control problem the knowledge about the dynamics is essential as it allows to estimate the (future) influence of a certain action. Therefore an accurate and robust system, i.e., state space, identification is an important precondition for an optimal control, i.e., for a real-world application of RL methods.

It has been shown that RNN of the form

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

can approximate any open dynamical system (eq. 1) and are therefore a good method for (complex and high-dimensional) system identification [10]. Here, the state transition equation s_t (eq. 2) 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 (eq. 2), s_t, in general does not have the same dimension as the one of the original open dynamical system (eq. 1) s_t^d. It basically depends on the system's complexity and the desired information accuracy.

The approximation of an open dynamical system (eq. 1) by an RNN of equation 2 is done by 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. equation 3, by adjusting the networks weight matrices A, B, and C as well as the bias θ.

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

This can be achieved 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 ($\tau = t, t-1, t-2, ..., t-m$, where m is the number of unfolded time steps) and for every pattern t ($t \geq m$) [2], [9]. It guarantees that we have the same dynamics in every unfolded time step τ. By using unfolding in time the RNN can be trained with a shared weights extension of the standard error backpropagation algorithm [11].

In its simplest form RNN unfolded in time only provide a one step prediction of the variables of interest, y_{t+1}^d. With regard to RL, i.e. an optimal state space identification, this is generally insufficient. For that 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 n time steps in future direction and use the network's own predictions as future inputs [1], [2]. By overshooting we increase the system approximation capability 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 [1].

Dynamical consistency solves the problem of the unavailable external information x_τ^d 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 performances. In RNN with dynamically consistent overshooting we therefore use the network's own predictions as a replacement for the unknown future inputs. This implies that we do not only forecast the variables of interest y_τ^d but all environment data x_τ^d. 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]:

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

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.

Besides their capability to approximate non-linear dynamical systems, RNN have the advantage that by using, in contrast to, e.g., Radial Basis Function (RBF) Networks and Support Vector Machines (SVMs) with local kernels,

[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 [2], [5].

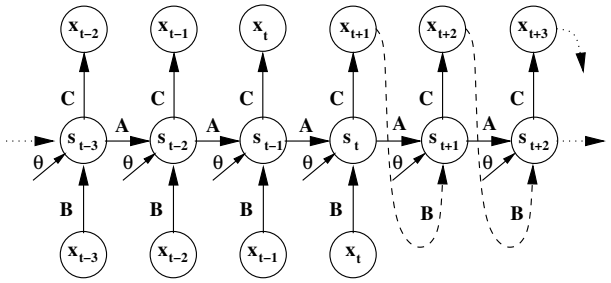


Fig. 1. RNN with dynamically consistent overshooting

global, sigmoidal basis functions, they are well able to cope with higher dimensions. Therefore they are suited to break the curse of dimensionality [12].

Considering optimal control we use the RNN to identify the system's minimal, approximately Markovian state space out of the observed high-dimensional data. We provide the RNN with all observables at each time step t as inputs and respectively targets x_τ^d and identify the underlying system. Here, the dimension of the inner state s_τ of the RNN is set to a desired value, which is minimal but sufficient to learn the problem's dynamics. In doing so, we can compress the observable's system information to the problem's essential state space. The dimension of s_τ just has to be large enough to evolve the original system development. In a way, we can say that we use the RNN for a dynamical system feature selection. Still, it is not a feature selection in the general meaning as we always take all available information into account and do not extract the most important ones. This has the particular advantage that the dimension reduction, i.e. variable compression, is done in an information conserving way. This also distinguishes our approach from dimension reductions with Feedforward Neural Networks, where, out of construction, one cannot incorporate the system's development.

Having a lower dimensional model of the regarded problem we can apply standard and well-known RL algorithms like Prioritised Sweeping [3], which only work for small dimensions, to determine an optimal policy.

Note, that a similar approach has been already validated for an artificial partially observable problem [13]. Here, the RNN was applied for a reconstruction of the (low dimensional) partially observable state space, which was again used as a basis for standard RL methods.

III. Recurrent Control Neural Network

The Recurrent Control Neural Network (RCNN) [5] is a Recurrent Neural Network (sec. II), which is able to identify and to control the dynamics of an RL or optimal control problem within one integrated network. 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 solving the complete optimal control problem within one network, i.e., the system identification as described in section II and the learning of the optimal policy, which has so far be done by standard RL methods. This has the advantage that in comparison to our RNN approach of section II we have one single network architecture instead of applying consecutively two different methods.

As the RCNN has to fulfill those two different tasks it is trained in two successive steps. Note, that this distinguishes our approach from other work on RL and Recurrent Neural Networks, e.g. [14], where one usually tries a combined learning of both tasks in one step. It has the advantage that, like in our approach of section II, 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 2.

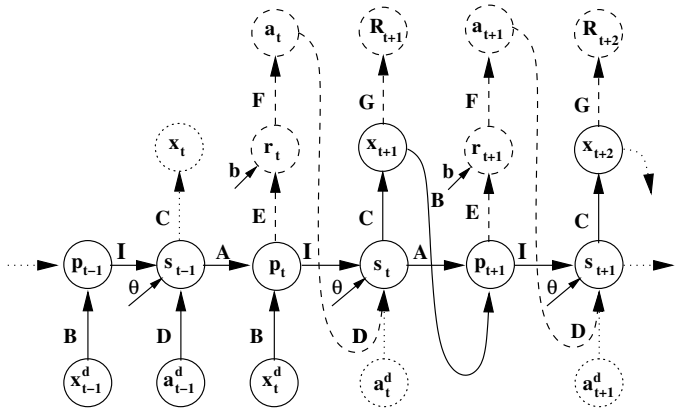


Fig. 2. Learning the optimal policy by a Recurrent Control Neural Network (RCNN). Dotted connections are just used in the first step whereas the dashed parts, the control networks, are only applied in the second one. In phase 1 matrices A, B, C, and D, coding the dynamics, are trained. In phase 2 the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

In the 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 (fig. 2, bold and dotted connections). Hence, analogue to those RNN (eq. 4), the optimisation task of step one takes on the following form:

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

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

In addition to the standard RNN (eq. 4) a pre-state p_τ is inserted, which serves in the second step as an input for the control network. Further, in contrast to the simpler RNN approach (sec. II) we separate the observable data into the environmental and the action variables, x_τ^d and

a^d_τ. This also requires the inclusion of an additional weight matrix D. In doing so, we get a mapping of the underlying Markov Decision Process on the input side [5]. Thereby it is important to note, that in this step the actions of the observed training data a^d_τ are also given to the RCNN as present and future inputs ($\tau \geq t$) because they directly influence the system's dynamics but cannot and should not be learned by the network (fig. 2, dotted connections).

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. 2, 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:

$$a_\tau \quad = \quad f(F\tanh(Ep_\tau + b)) \quad \forall \tau \geq t \tag{6}$$

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 network's action space. The hidden state (fig. 2) of the control network is denoted by r_τ.

As we want to determine the optimal action a_τ, the control network (eq. 6) 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. 2, 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 past unfolding ($\tau < t$) the output-clusters are taken away, because they are only needed for the identification of the system dynamics. In the present and future part ($\tau \geq t$) of the network the error-function (eq. 5) 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 as well as a possible activation function h within the output clusters x_τ (fig. 2). In other words the RCNN maps the reward function $c(\cdot)$ of the underlying RL problem by coding it in a neural architecture, which, as in the case of the gas turbine control (sec. IV), often also requires some additional, fixed connected, clusters.

The weights of the control network are only adapted according to the backpropagated 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 given the system dynamics modelled in step one (eq. 5). 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:

$$\begin{aligned}
s_\tau &= \begin{cases} \tanh(\mathbf{I}p_\tau + Da^d_\tau + \theta) & \forall \tau < t \\ \tanh(\mathbf{I}p_\tau + Da_\tau + \theta) & \forall \tau \geq t \end{cases} \\
R_{\tau+1} &= Gh(Cs_\tau), \quad \forall \tau \geq t \\
\text{with } a_\tau &= f(F\tanh(Ep_\tau + b)) \quad \forall \tau \geq t \\
\text{and } p_\tau &= \begin{cases} As_{\tau-1} + Bx^d_\tau & \forall \tau \leq t \\ As_{\tau-1} + Bx_\tau & \forall \tau > t \end{cases} \\
\sum_t \sum_{\tau \geq t} c(R_\tau) &\rightarrow \min_{E,F,b}
\end{aligned} \tag{7}$$

In both steps (eqs. 5 and 7) the RCNN is trained on the identical set of training patterns T and with backpropagation [11]. 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 (sec. IV) or reconstruct a partially observable environment [13]. Furthermore, out of the construction of the RCNN, it can well handle continuous state and action spaces.

IV. Controlling a gas turbine simulation

The data, we worked on, was taken from a gas turbine as it is used for electrical power generation. In total, approximately 1.3 million real data samples were available. The time spans covered by those included several months and different operating points of the turbine. The overall objective was to optimise the turbine's operation for stable combustion on high load.

Out of the available data we developed a simulation model of the real turbine, as experiments on the real turbine were too expensive. This was done with the help of a dedicated Neural Network architecture but, for an easier comprehension and due to the lack of space, we abstain from a detailed description. The simulation operates on 20 different parameters such as electrical load (power), exhaust temperature, temperature and pressure behind the compressor, fuel fractions, acceleration and pressure intensities in the combustion chamber (RMS). Their dynamic ranges were taken from the original data. Regarded actions were limited modifications of pilot gas and inlet guide vane (IGV), which are two master control parameters of such turbines. The time grid was set to 5 seconds, i.e., the simulation model was used to predict the state of the turbine 5 seconds ahead.

The simulation served as basis for testing our two Neural RL approaches (sec. II and III). In order to guarantee the quality of our experiments we first checked the accuracy of our simulation by comparing the model's one-step prediction

of all data points with the true behaviour of the gas turbine. We further tested the influence of different control parameters and verified their physical plausibility, to evaluate if a direct interaction with the turbine is feasible. The experiments have shown that the simulation well captures the important aspects of the gas turbine's dynamics as we achieved a relative approximation error of less than 2.6%. Moreover the closed loop iteration over longer time periods showed a stable behaviour that keeps the parameters in a valid range.

Although we already made the problem more tractable by working on a simulation, a direct control by discrete and localised RL methods, such as Q-Learning or Temporal-Difference-Learning on local basis functions, appeared to be infeasible, as the state space was still of high dimension. Further the state and action spaces were both continues. Therefore, we applied our two new neural-based methods. For both we allowed 100.000 observations for training and tested the learned policy afterwards on the simulation. The reward function used was composed out of a scaled difference between the parameters electrical load (power) as well as the acceleration and pressure intensities in the combustion chamber (RMS).

A. RNN Approach

We applied our RNN approach as described in section II. Consequently we used an RNN to reduce the high-dimensional state space of the turbine simulation to make it applicable for standard table-based RL methods on a sufficiently fine-gridded discretisation, for which convergence to an optimal solution can be guaranteed.

The RNN was provided with the 20 dimensional state space of the simulation as inputs and respectively targets x_τ^d. Even though the simulation operates on 20 variables, its dynamics requires embedding in time and thus multiplies the number of required dimensions. Still, the idea is to condense this state description into a lower dimensional, but approximately Markovian state, on which table-based RL algorithms can be used. We started with a 20 dimensional internal state space s_τ of the RNN and incrementally applied node pruning [15] to it. In doing so we achieved a sufficiently acceptable generalisation performance with an only four dimensional state space, which served as a basis for Prioritized Sweeping (PS) [3] with ϵ-greedy exploration to determine the optimal policy.

B. RCNN

As a further improvement, we applied the RCNN (sec. III), incorporating the two steps, dimension reduction and RL, in one network. Analogue to the RNN approach (sec. IV-A), the RCNN also used the 20 dimensional state space of the simulation as inputs and targets, x_τ^d. It further got the two control parameters provided as actions a_τ^d. These were also the ones determined by the control network in the second learning step (eq. 6). Here, the hyperbolic tangens was used as output activation function f, which limits the actions to $(-1, 1)$. To further constrain the influence of the calculated actions an additional cluster with fixed connections and weight values smaller than one was added. The reward function was (hard) coded into the neural architecture. Note, that as a preprocessing the parameters were all scaled to the interval $[-1, 1]$ but were re-scaled for the reward calculation, as this made the computations more accurate. The RCNN is unfolded 8 steps into the past and 24 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 100 and the hidden state of the control network, $\dim(r)$, to 20 neurons. These dimensions are effectual to generate stable results in terms of system identification and learning the optimal policy.

C. Results

The results (fig. 3) clearly show that our neural RL methods, RNN approach in combination with ϵ-greedy PS (dashed line) and the RCNN (solid line), outperform the reference controller (dotted-dashed line), whose behaviour is reflected by the original turbine data, by far. They also achieve higher rewards than a simple controller based on ϵ-greedy PS on a well selected sub-space (dotted line), which we used as an additional benchmark. This also demonstrates the high value of a minimal state space identification by an RNN. It turns out, that the RCNN reaches the highest reward, which underlines the advantages of this integrated approach.

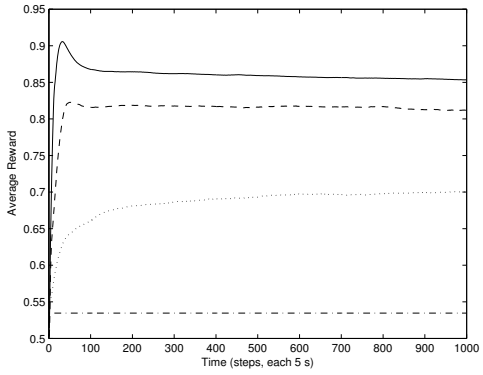
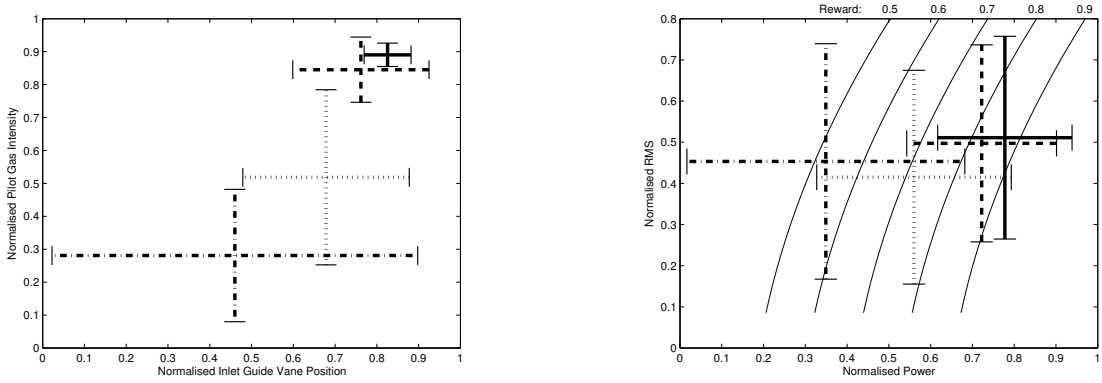


Fig. 3. Development of the reward for RCNN (solid), RNN with ϵ-greedy PS (dashed), simple ϵ-greedy PS (dotted) and the reference controller (dotted-dashed) in time.

Figure 4 illustrates the performance behaviour of the four methods after reaching a stable operating point. Note, that the values are normalised for confidentiality reasons. Figure 4(a) compares the respective mean setting of the control parameters, pilot gas and IGV. It shows that the RCNN (solid line) developed the most robust and stable policy as its standard deviation is quite low. Figure 4(b) depicts the two major performance indicators, power and RMS. It confirms the good results of the RCNN (solid line) as it reaches the highest power level with only a minor increase in RMS. For a better presentation we plotted "iso-reward" curves, which indicate identical reward for different settings.



(a) Control parameters pilot gas and IGV

(b) Performance indicators power and RMS

Fig. 4. Mean value and standard deviation for (a) the setting of the two control parameters and (b) the two major performance indicators for the RCNN (solid), RNN with ϵ-greedy PS (dashed), simple ϵ-greedy PS (dotted) and the reference controller (dotted-dashed), after reaching a stable operation point.

V. Conclusion and Outlook

We presented a new Neural Reinforcement Learning approach to an optimal gas-turbine control and showed its capability to significantly improve the turbine's stability and its lifetime by guaranteeing the accustomed high performance.

We started with a short introduction into RNN, which we used for a minimal state space identification of the regarded high-dimensional problem. This made it tractable for standard RL methods, which generally only work in low dimensions. We proceeded by integrating the two step RNN approach into one single so called Recurrent Control Neural Network (RCNN), which is, due to its combination of an RNN and a three layer Neural Network, well qualified for solving high-dimensional optimal control problems with continuous state and action spaces.

We finally applied our two methods to the simulation of a gas turbine. The results were promising as with both methods we outperformed standard controllers by far. It became evident that the integrated structure of the RCNN approach is more powerful then the two step RNN one. Nevertheless the latter showed despite its simplicity a remarkable performance.

Future work is directed towards the control of a real gas turbine. Besides that an improvement of the RCNN architecture, in particular a reduction or aggregation of the several different matrices, and an extension of the method to online-learning is projected. Here, we will also refer to higher developed Neural Network architectures like Dynamical Consistent Neural Networks [2] and incorporate several other neural methods, e.g. pruning, or monte carlo techniques.

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] 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.

[2] 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.

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

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

[5] 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, in preparation.

[6] Q. Song, "An integrated robust/neural controller with gas turbine applications," in *IEEE Conference on Control Applications*, 1994.

[7] N. W. Chbat, "Estimating gas turbine internal cycle parameters using a neural network," *International Gas Turbine and Aeroegine Congress and Exhibition*, 1996.

[8] I. T. Nabney and D. C. Cressy, "Neural network control of a gas turbine," *Neural Computing and Applications*, vol. 4, no. 4, 1996.

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

[10] 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.

[11] 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.

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

[13] 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.

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

[15] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995.