Dynamical Consistent Recurrent Neural Networks

Hans-Georg Zimmermann, Ralph Grothmann, Anton M. Schäfer and Christoph Tietz
Siemens AG, Corporate Technology, Information & Communication Division, Dept. of Neural Computation
Hans_Georg.Zimmermann@siemens.com

***Abstract*— Recurrent neural networks are typically considered as relatively simple architectures, which come along with complicated learning algorithms. Most researchers focus on the improvement of these algorithms. Our approach is different: Rather than focusing on learning and optimization algorithms, we concentrate on the design of the network architecture.**

As we will show, many difficulties in the modeling of dynamical systems can be solved with a pre-design of the network architecture. We will focus on large networks with the task of modeling complete high dimensional systems (e.g. financial markets) instead of small sets of time series. Standard neural networks tend to overfit like any other statistical learning system. We will introduce a new recurrent neural network architecture in which overfitting and the associated loss of generalization abilities is not a major problem. We will enhance these networks by dynamical consistency.

I. Introduction

Recurrent neural networks allow the identification of dynamical systems in form of high dimensional, nonlinear state space models. They offer an explicit modeling of time and memory and allow in principle to model any type of dynamical systems [Hay94; MJ99; KK01]. The basic concept is as old as the theory of artificial neural networks, so e.g. unfolding in time of neural networks and related modifications of the back-propagation algorithm can be found in [Wer74] and [RHW86]. Different types of learning algorithms are summarized in the paper of Pearlmutter [Pearl95]. Nevertheless, over the last 15 years most feedforward neural networks have been predominantly applied to time series problems. The appealing way of modeling time and memory in recurrent networks is opposed to the apparent easier numerical tractability of a pattern recognition approach as represented by feedforward neural networks. Still some researchers enhance the theory of recurrent neural networks. Recent developments are summarized in the books of Haykin [Hay94], Kolen and Kremer [KK01], Soofi and Cao [SC02] and Medsker and Jain [MJ99].

Our approach differs from the outlined research directions in a significant but at first sight non-obvious way. Instead of focusing on algorithms, we put network architectures in the foreground. We show, that the design of a network architecture automatically implies the usage of an adjoint solution algorithm for the parameter identification problem. This correspondence between architecture and equations is valid for simple as well as complex network architectures. The underlying assumption is, that the associated parameter optimization problem is solved by error backpropagation through time, i.e. a shared weights extension of the standard error backpropagation algorithm.

In our previous papers we discussed the modeling of dynamical systems based on time-delay recurrent neural networks [NZ98; ZN01]. We solved the system identification task by unfolding in time, i. e. we transferred the temporal problem into a spatial architecture, which can be handled by the error backpropagation through time [RHW86; Hay94, p. 354-357 and p. 751-756]. Proceeding this way, we can enforce the learning of the autonomous dynamics in an open system by overshooting [ZN01, p.326-327]. Consequently our recurrent neural networks not only learn from data but also integrate prior knowledge and first principles into the modeling in form of architectural concepts.

However, the question arises if the outlined neural networks are a sufficient framework for the modeling of complex nonlinear and high dimensional dynamical systems, which can only be understood by analyzing the interrelationship of different sub-dynamics. Our experiments indicate, that simply scaling up the basic time-delay recurrent neural networks by increasing the dimension of the internal state, results in overfitting due to the high number of free parameters.

In this paper we present architectures which are feasible for large recurrent neural networks. These architectures are based on a redesign of the basic time-delay recurrent neural networks [ZN01; Hay94, p. 322-323 and p. 739-747]. Remarkably, most of the resulting networks cannot even be designed with a low dimensional internal state (see sec. II). We further move from an often arbitrary distinction between input and output to a modeling of observables (see sec. III). In addition, we focus on a consistency problem of traditional statistical modeling: Typically one assumes, that the environment of the system remains unchanged when the dynamics is iterated into the future direction. We show, that this is a questionable statistical assumption and solve the problem with a dynamical consistent recurrent neural network (see sec. IV).

II. Normalization of Recurrent Networks

For discrete time grids a basic time-delay recurrent neural network can be described with a state transition equation s_t and output equation y_t:

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

The state transition equation s_t is a nonlinear combination of the previous state s_{t-1} and external influences u_t using weight matrices A and B and a bias c, which handles offsets in the input variables u_t. The network output y_t is computed from the present state s_t employing matrix C. The network output is therefore a nonlinear composition applying the transformations A, B and C [ZN01; Hay94, p. 322-323 and p. 739-747]:

As a preparation for the development of large networks we first separate the state equation of the basic time-delay recurrent network (Eq. 1) into a past and a future part. In this framework s_t is always regarded as the present time state. All states s_τ with $\tau \leq t$ belong to the past part and those with $\tau > t$ to the future part. The parameter τ is hereby always bounded by the length of the unfolding in time m and the length of the overshooting n (see [ZN01; ZNG02]). We have $\tau \in \{t-m, \ldots, t+n\}$ for all $t \in \{m, \ldots, T-n\}$ with T as the total number of available data patterns. The present time ($\tau = t$) is included in the past part, as these state transitions share the same characteristics. We get the following representation of the optimization problem:

$$\begin{aligned} \tau \leq t: \quad s_{\tau+1} &= \tanh(As_\tau + c + Bu_\tau) \\ \tau > t: \quad s_{\tau+1} &= \tanh(As_\tau + c) \\ y_\tau &= Cs_\tau \\ \sum_{t=m}^{T-n} \sum_{\tau=t-m}^{t+n} (y_\tau - y_\tau^d)^2 &\to \min_{A,B,C,c} \end{aligned} \tag{2}$$

Using finite unfolding in time, these equations can be easily transformed into a neural network architecture (see Fig. 1).

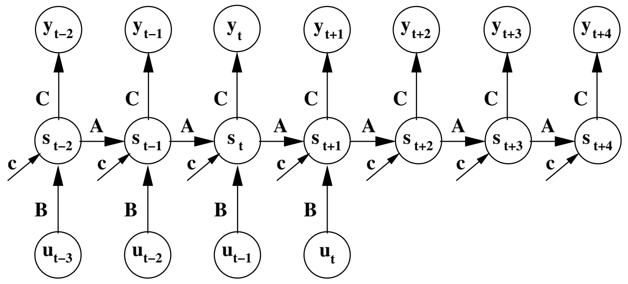

Fig. 1. Unfolded recurrent neural network.

In this model, past and future iterations are consistent under the assumption of a constant future environment. The difficulty with this kind of recurrent neural network is the training with backpropagation through time, because a sequence of different connectors has to be balanced. The gradient computation is not regular, i. e. we do not have the same learning behavior for the weight matrices in the different time steps. Even the training itself is unstable due to the concatenated matrices A, B and C. As the training changes weights in all of these matrices, different effects or tendencies – even opposing ones – can influence them and may superpose. This implies, that there results no clear learning direction or change of weights from a certain backpropagated error. In our experiments we found, that these problems become even more important for the training of large recurrent neural networks.

Now the question arises, how to re-design the basic recurrent architecture (Eq. 2), such that the learning behavior and the stability improves especially for large networks.

As a remedy, we propose the neural network of Eq. 3 which incorporates besides the bias c only one connector type, the matrix A. The resulting architecture is depicted in Fig. 2.

$$\begin{aligned} \tau \leq t: \quad s_\tau &= \tanh\left(As_{\tau-1} + c + \begin{bmatrix} 0 \\ 0 \\ Id \end{bmatrix} u_\tau\right) \\ \tau > t: \quad s_\tau &= \tanh(As_{\tau-1} + c) \\ y_\tau &= [Id\ 0\ 0] s_\tau \\ \sum_{t=m}^{T-n} \sum_{\tau=t-m}^{t+n} (y_\tau - y_\tau^d)^2 &\to \min_{A,c} \end{aligned} \tag{3}$$

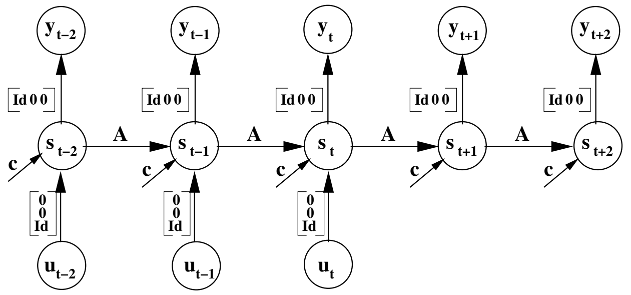

Fig. 2. Normalized recurrent neural network.

We call this model a Normalized Recurrent Neural Network. It avoids the stability and learning problems resulting from the concatenation of the three matrices A, B and C. The modeling is now solely focused on the transition matrix A. The matrices between input and hidden as well as hidden and output layer are fixed, i. e. they are not learned. This implies that all free parameters – as they are combined in one matrix – are now treated the same way by error backpropagation.

It is important to note, that the normalization or concentration on only one single matrix is paid with an oversized (high dimensional) internal state. At first view it seems, that in this network architecture (Fig. 2) the external input u_τ is directly connected to the corresponding output y_τ. This is not the case, because we enlarge the dimension of the internal state s_τ, such that the input u_τ has no direct influence on the output y_τ. Assuming that we have a number of p network outputs, q computational hidden neurons and r external inputs, the dimension of the internal state would be $\dim(s) = p+q+r$.

With the matrix $[Id\ 0\ 0]$ we connect only the first p neurons of the internal state s_τ to the output layer y_τ. As this connector is not trained, it can be seen as a fixed identity matrix of appropriate size. Consequently, the neural network is forced to generate the p outputs of the neural network at the first p components of the state vector s_τ.

Let us now focus on the last r state neurons, which are used for the processing of the external inputs u_τ. The connector $[0\ 0\ Id]^T$ between the externals u_τ and the internal state s_τ is an appropriately sized fixed identity matrix. More precisely, the connector is designed such that the input u_τ is connected to the last state neurons. Recalling that the network outputs are located at the first p internal states, this composition avoids a direct connection between input and output. It delays the impact of the externals u_τ on the outputs y_τ by at least one time step.

To additionally support the internal processing and to increase the network's computational power, we add a number of q hidden neurons between the first p and the last r state neurons. This composition ensures, that the input and output processing of the network is separated.

Besides the bias vector c the state transition matrix A holds the only tunable parameters of the system. Matrix A does not only code the autonomous and the externally driven part of the dynamics, but also the processing of the external inputs u_τ and the computation of the network outputs y_τ.

Most remarkably, the normalized recurrent neural network of Eq. 3 can only be designed as a large neural network. If the internal state of the network is too small, the inputs and outputs can not be separated, as the external inputs would at least partially cover the internal states at which the outputs are read out. Thus, the identification of the network outputs at the first p internal states would become impossible.

Our experiments indicate, that recurrent neural networks in which the only tunable parameters are located in a single state transition matrix (e.g. Eq. 3) show a more stable training behavior, even if the dimension of the internal state is very large. Having trained the large network to convergence, many weights of the state transition matrix will be dispensable without affecting the functioning of the network. Unneeded weights can be singled out by using a weight decay penalty and standard pruning techniques (see e. g. [Hay94; NZ98]).

III. Modeling the dynamics of observables

In the normalized recurrent neural network (Eq. 3) we consider inputs and outputs independently. This distinction between externals u_τ and the network output y_τ is arbitrary and mainly depends on the application or the view of the model builder instead of the true underlying dynamical system. Therefore, for the following model we take a different point of view. We merge inputs and targets into one group of variables, which we call observables. So we now look at the model as a high dimensional dynamical system where input and output represent the observable variables of the environment. The hidden units stand for the unobservable part of the environment, which nevertheless can be reconstructed from the observations. By doing so, we arrive at an integrated view of the dynamical system.

We implement this approach by replacing the externals u_τ with the (observable) targets y_τ^d in the normalized recurrent network. Consequently, the output y_τ and the external input y_τ^d have now identical dimensions.

$$\begin{aligned} \tau \le t: \quad & s_\tau = \tanh\left(As_{\tau-1} + c + \begin{bmatrix} 0 \\ 0 \\ Id \end{bmatrix} y_\tau^d\right) \\ \tau > t: \quad & s_\tau = \tanh(As_{\tau-1} + c) \\ & y_\tau = [Id\ 0\ 0] s_\tau \\ & \sum_{t=m}^{T-n} \sum_{\tau=t-m}^{t+n} (y_\tau - y_\tau^d)^2 \to \min_{A,c} \end{aligned} \tag{4}$$

The corresponding model architecture (Fig. 3) changes only slightly in comparison to Fig. 2.

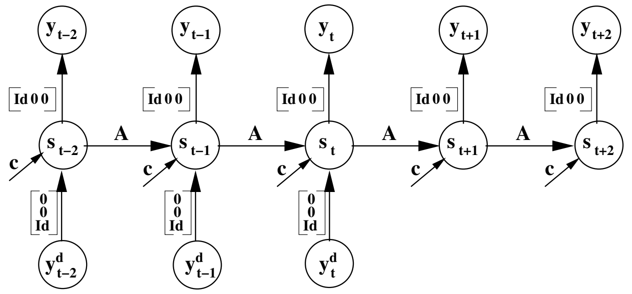

Fig. 3. Normalized recurrent neural network modeling the dynamics of observables y_τ^d.

Note, that due to the one step time delay between input and output, y_τ^d and y_τ are not directly connected. Furthermore it is important to understand, that we now take a totally different view on the dynamical system. In contrast to Eq. 3, the network (Eq. 4) not only generates forecasts for the dynamics of interest but also for all external observables y_τ^d. Consequently, the first r state neurons are used for the identification of the network outputs. They are followed by q computational hidden neurons and r state neurons which read in the external inputs.

IV. Dynamical Consistent Neural Networks

An open dynamical system is partially driven by an autonomous development and partially by external influences. If the dynamics is iterated into the future, the development of the system environment is unknown. Now, one of the standard statistical paradigms is to assume, that the external influences are not significantly changing in the future. This means, that the expected value of a shift in an external input y_τ^d with $\tau > t$ is zero per definition. For that reason we have so far neglected the external inputs y_τ^d in the normalized recurrent neural network at all future time steps, $\tau > t$, of the unfolding (see Eq. 4).

Especially when we consider fast changing external variables with a high impact on the dynamics of interest, the latter assumption is very questionable. In relation to Eq. 4 it even poses a contradiction, as the observables are assumed to be constant on the input, but fluctuate on the output side. Even in case of a slowly changing environment, long-term forecasts become doubtful. The longer the forecast horizon is, the more the statistical assumption of a constant environment is violated. A statistical model is therefore not consistent from a dynamical point of view. For a dynamical consistent approach, one has to integrate assumptions about the future development of the environment into the modeling of the dynamics.

For that reason we propose a network that uses its own predictions as replacements for the unknown future observables. The resulting dynamical consistent recurrent neural network, DCRNN, is specified in Eq. 5:

$$\begin{aligned}
\tau \le t: \quad s_\tau &= \begin{bmatrix} Id & 0 & 0 \\ 0 & Id & 0 \\ 0 & 0 & 0 \end{bmatrix} \tanh(As_{\tau-1}+c) + \begin{bmatrix} 0 \\ 0 \\ Id \end{bmatrix} y_\tau^d \\
\tau > t: \quad s_\tau &= \begin{bmatrix} Id & 0 & 0 \\ 0 & Id & 0 \\ Id & 0 & 0 \end{bmatrix} \tanh(As_{\tau-1}+c) \\
y_\tau &= [Id\; 0\; 0] \quad s_\tau \\
\sum_{t=m}^{T-n} \sum_{\tau=t-m}^{t+n} & (y_\tau - y_\tau^d)^2 \to \min_{A,c}
\end{aligned} \tag{5}$$

The DCRNN is understood best by analyzing the structure of the state vector s_τ. In the past ($\tau \le t$) and future ($\tau > t$) the structure of the internal state s_τ is

$$s_\tau = \begin{bmatrix} y_\tau \\ h_\tau \\ \left\{ \begin{array}{l} \tau \le t: \; y_\tau^d \\ \tau > t: \; y_\tau \end{array} \right\} \end{bmatrix} = \begin{bmatrix} \text{expectations} \\ \text{hidden states} \\ \left\{ \begin{array}{l} \tau \le t: \; \text{observations} \\ \tau > t: \; \text{expectations} \end{array} \right\} \end{bmatrix} \tag{6}$$

In the first r components of the state vector we have the expectations y_τ, i.e. the predictions of the model. The q components in the middle of the vector represent the hidden units h_τ. They are responsible for the development of the dynamics. In the last r components of the vector we find in the past ($\tau \le t$) the observables y_τ^d, which the model receives as external input. In the future ($\tau > t$) the model replaces the unknown future observables by its own expectations y_τ. This replacement is modeled with two consistency matrices:

$$C_\le = \begin{bmatrix} Id & 0 & 0 \\ 0 & Id & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ and } C_> = \begin{bmatrix} Id & 0 & 0 \\ 0 & Id & 0 \\ Id & 0 & 0 \end{bmatrix} \tag{7}$$

Let us explain one recursion of the state equation (Eq. 5) step by step: In the past ($\tau \le t$) we start with a state vector $s_{\tau-1}$, which has the structure of Eq. 6. This vector is first multiplied with the transition matrix A. After adding the bias c, the vector is sent through the nonlinearity $\tanh$. The consistency matrix then keeps the first $r+q$ components (expectations and hidden states) of the state vector but deletes (multiplication with zero) the last r ones. These are finally replaced by the observables y_τ^d, such that s_τ has again the partitioning of Eq. 6. Note that in contrast to the normalized recurrent neural network (Eq. 4) the observables are now added to the state vector after the nonlinearity. This is important for the consistency structure of the model.

The recursion of the future state transition ($\tau > t$) differs from the one in the past in terms of the structure of the consistency matrix and the missing external input. The latter is now replaced with an additional identity-block in the future consistency matrix $C_>$ which maps the first r components of the state vector, the expectations y_τ, to its last r components. In doing so we get the desired partitioning of s_τ (Eq. 6) and the model becomes dynamical consistent.

Fig. 4 illustrates the corresponding architecture. Note, that the nonlinearity and the final calculation of the state vector are separated and hence modeled in two different layers. This follows from the dynamical consistent state equation (Eq. 5), in which the observables are added separate from the nonlinear component.

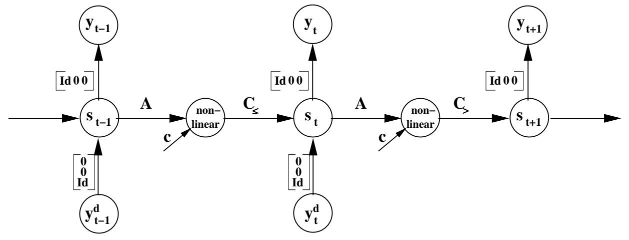

Fig. 4. Dynamical Consistent Recurrent Neural Network (DCRNN). At all future time steps of the unfolding the network uses its own forecasts as substitutes for the unknown development of the environment.

Regarding the transition matrix A, we want to point out that in a statistical consistent recurrent network (Eq. 4) the matrix has to model the state transformation over time and the merging of the input information. However, the network is only triggered by the external drivers up to the present time step t. In a dynamical consistent network we have forecasts of the external influences, which can be used as future inputs. Thus, the transition matrix A is always dedicated to the same task: modeling the dynamics.

V. Conclusions

In this article we focused on high-dimensional and dynamical consistent recurrent neural networks for the modeling of open dynamical systems. These networks allow an integrated view on real-world problems and consequently show better generalization abilities. We concentrated the modeling of the dynamics on one single transition matrix and also enhanced from a simple statistical to a dynamical consistent handling of missing input information in the future part. The networks are now able to map integrated dynamical systems (e. g. coherent financial markets) instead of only a small set of time series. Remarkably, these recurrent neural networks do not only provide superior forecasts, but also a deeper understanding of the underlying dynamical system.

Current applications for dynamical consistent neural networks are financial and commodity market price forecasts.

Further research is done on a combination of dynamical consistency and error correction neural networks, ECNN [ZNG02]. ECNN use an error correction mechanism for a quantification of the model's misfit and as indicator of short-term effects or external shocks. We expect that dynamical consistent recurrent neural networks with error correction will further improve the ability of the identification and forecasting of complex and high-dimensional dynamical systems.

Acknowledgment

All computations are performed with the *Simulation Environment for Neural Networks (SENN)*, a product of Siemens AG.

References

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

[HSW92] Hornik K., Stinchcombe M. and White H.: *Multilayer Feedforward Networks are Universal Approximators*, in: White H. et al. [Eds.]: Artificial Neural Networks : Approximation and Learning Theory, Blackwell Publishers, Cambridge, MA, 1992.

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

[MC01] Mandic D. P. and Chambers J. A.: *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*, Wiley Series in Adaptive Learning Systems for Signal Processing, Communications & Control, S. Haykin [Ed.], J. Wiley & Sons, Chichester 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.

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

[Pearl95] Pearlmatter B.: *Gradient Calculations for Dynamic Recurrent Neural Networks: A survey*, in: IEEE Transactions on Neural Networks, Vol. 6, Nr. 5, 1995.

[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: M.I.T. Press, pp. 318-362, 1986.

[SC02] Soofi, A. and Cao, L.: *Modeling and Forecasting Financial Data, Techniques of Nonlinear Dynamics*, Kluwer Academic Publishers, 2002.

[Wer74] Werbos P. J.: *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD. Thesis, Harvard University, 1974.

[ZGT02] Zimmermann H. G., Grothmann R. and Tietz Ch.: *Yield Curve Forecasting by Error Correction Neural Networks and Partial Learning*, in: M. Verleysen [Ed.], Proceedings of the European Symposium on Artificial Neural Networks (ESANN) 2002, Bruges, Belgium, pp. 407-12.

[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

[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, 2001, pp. 311-350.

[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, 2002.