Forecasting with Recurrent Neural Networks: 12 Tricks

Hans-Georg Zimmermann, Christoph Tietz and Ralph Grothmann

Siemens AG, Corporate Technology
Otto-Hahn-Ring 6, D-81739 Munich, Germany
Hans_Georg.Zimmermann@siemens.com

Abstract. Recurrent neural networks (RNNs) are typically considered as relatively simple architectures, which come along with complicated learning algorithms. This paper has a different view: We start from the fact that RNNs can model any high dimensional, nonlinear dynamical system. Rather than focusing on learning algorithms, we concentrate on the design of network architectures. Unfolding in time is a well-known example of this modeling philosophy. Here a temporal algorithm is transferred into an architectural framework such that the learning can be performed by an extension of standard error backpropagation.
We introduce *12* tricks that not only provide deeper insights in the functioning of RNNs but also improve the identification of underlying dynamical system from data.

1 Introduction

In many business management disciplines, complex planning and decision-making can be supported by quantitative forecast models that take into account a wide range of influencing factors with non-linear cause and effect relationships. Furthermore, the uncertainty in forecasting should be considered. The procurement of raw materials or demand planning are prime examples: The timing of copper purchases can be optimized with accurate market price forecasts, whereas precise forecasts of product sales increase the deliver reliability and reduce costs. Likewise technical applications, e. g. energy generation, also require the modeling of complex dynamical systems.

In this contribution we deal with time-delay recurrent neural networks (RNNs) for time series forecasting and introduce *12* tricks that not only ease the handling of RNNs, but also improve the forecast accuracy. The RNNs and associated tricks are applied in many of our customer projects from economics and industry.

RNNs offer significant benefits for dealing with the typical challenges associated with forecasting. With their universal approximation properties [11], RNNs can model high-dimensional, non-linear relationships. The time-delayed information processing addresses temporal structures. In contrast, conventional econometrics generally uses linear models (e.g. autoregressive models (AR), multivariate linear regression) which can be efficiently estimated from historical data, but provide only an inadequate framework for non-linear dynamical systems [12].

In Section 2 we introduce the so-called correspondence principle for neural networks (Trick *1*). For us neural networks are a class of functions which can be transformed into architectures. We will work only with algorithms that process information locally within the architectures. As we will outline, for some problems it is easier to start off with the NN architecture and formulate the equations afterwards and for other problems vice versa. The locality of the algorithms enables us to model even large systems. The correspondence principle is the basis for different RNN models and associated tricks.

We will start with a basic RNN in state space formulation for the modeling of partly autonomous and partly externally driven dynamical systems (so-called open systems). The associated parameter optimization task is solved by (finite) unfolding in time, which can be handled by a shared weights extension of standard backpropagation. Dealing with state space models, we are able to utilize memory effects. Therefore, there is no need of a complicated input preprocessing in order to represent temporal relationships. Nevertheless, learning of open dynamical system tends to focus on the external drivers and, thus, neglects the identification of the autonomous part. On this problem Trick *2* enforces the autonomous flow of the dynamics and thus, enables long-term forecasting. Trick *3* finds a proper initialization for the first state vector of recurrent neural network in the finite unfolding.

Typically we do not know all external drivers of the open dynamical system. This may cause the identification of pseudo causalities. Trick *4* is the extension of the RNN with an error correction term, resulting in a so-called error correction neural network (ECNN), which enables us to handle missing information, hidden external factors or shocks on the dynamics. ECNNs are an appropriate framework for low-dimensional dynamical systems with less than 5 target variables. For the modeling of high-dimensional systems on low dimensional manifolds as in electrical load curves Trick *5* adds a coordinate transformation (so-called bottleneck) to the ECNN.

Standard RNNs use external drivers in the past and assume constant environmental conditions from present time on. For fast changing environments this is a questionable assumption. Internalizing the environment of the dynamics into the model, leads to Trick *6*, so-called historically consistent neural networks (HCNN). The special feature of the HCNN is that it not only models the individual dynamics of interest, but also models the external drivers. This leads to a closed dynamical system formulation. Therefore, HCNNs are symmetric in their input and output variables, i. e. the system description does not draw any distinction between input, output and internal state variables.

In practice, HCNNs are difficult to train, because the models have no input signals and are unfolded across the complete data horizon. This implies that we learn from a single data pattern, which is the unique data history. In Trick *7* we therefore introduce an architectural teacher forcing to make the best possible use of the data from the observables and to accelerate training of the HCNN.

The HCNN models the dynamics for all of the observables and their interaction in parallel. For this purpose a high-dimensional state transition matrix is required. A fully connected state transition matrix can, however, lead to a signal overload during the training of the neural network using error backpropagation through time (EBTT). With Trick *8* we solve this problem by introducing sparse state transition matrices.

The information flow within a HCNN is from the past to present and future time, i. e. we have a causal model to explain the highly-interacting non-linear dynamical systems across multiple time scales. Trick *9* extends this modeling framework with an information flow from the future into past. As we will show this enables us to incorporate the effects of rational decision making and planning into the modeling. The resulting models are called *causal-retro-causal* historically consistent neural networks (CRCNNs). Likewise to HCNNs, CRCNNs are difficult to train. In Trick *10* we extend the basic CRCNN architecture by an architectural teacher forcing mechanism, which allows us to learn the CRCNN using the standard EBTT algorithm. Trick *11* introduces a way to improve the modeling of deterministic chaotic systems.

Finally, Trick *12* is dedicated to the modeling of uncertainty and risk. We calculate ensembles of either HCNNs or CRCNNs to forecast probability distributions. Both modeling frameworks give a perfect description of the dynamic of the observables in the past. However, the partial observability of the world results in a non-unique reconstruction of the hidden variables and thus, different future scenarios. Since the genuine development of the dynamics is unknown and all paths have the same probability, the average of the ensemble is the best forecast, whereas the ensemble bandwidth describes the market risk.

Section 3 summarizes the primary findings of this contribution and points to future directions of research.

2 Tricks for Recurrent Neural Networks

Trick 1. The Correspondence Principle for Neural Networks

In order to gain a deeper understanding in the functioning and composition of RNNs we introduce our first conceptual trick, which is called correspondence principle between equations, architectures and local algorithms. The correspondence principle for neural networks (NN) implies that any equation for a NN can be portrayed in graphical form by means of an architecture which represents the individual layers of the network in the form of nodes and the matrices between the layers in the form of edges. This correspondence is most beneficial in combination with local optimization algorithms that provide the basis for the training of the NNs. For example, the error back propagation algorithm needs only locally available information from the forward and backward flow of the network in order to calculate the partial derivatives of the NN error function[13]. The use of local algorithms here provides an elegant basis for the expansion of the neural network towards the modeling of large systems. Used in combination with an appropriate (stochastic) learning rule, it is possible to use the gradients as a basis for the identification of robust minima[8].

Now let us introduce a basic recurrent neural network (RNN) in state space formulation. We start from the assumption that a vector time series y_τ is created by an open dynamical system, which can be described in discrete time τ using a state transition and output equation[3]:

$$\begin{array}{ll} \text{state transition} & s_{\tau+1} = f(s_\tau, u_\tau) \\ \text{output equation} & y_\tau \quad = g(s_\tau) \end{array} \tag{1}$$

The hidden time-recurrent state transition equation $s_{\tau+1} = f(s_\tau, u_\tau)$ describes the upcoming state $s_{\tau+1}$ by means of a function of the current system state s_τ and of the external factors u_τ. The system formulated in the state transition equation can therefore be interpreted as a partially autonomous and partially externally driven dynamic. We call this an open dynamical system.

The output, also called observer, equation y_τ uses the non-observable system state s_τ in every time step τ to calculate the output of the dynamic system y_τ. The data-driven system identification is based on the selected parameterized functions $f()$ and $g()$. We chose the parameters in $f()$ and $g()$ such that an appropriate error function is minimized (see Eq. 2).

$$\frac{1}{T}\sum_{\tau=1}^{T}\left(y_\tau - y_\tau^d\right)^2 \rightarrow \min_{f,g} \ . \tag{2}$$

The two functions $f()$ and $g()$ are estimated using the quadratic error function (Eq. 2) in such a way that the average distance between the observed data y_τ^d and the system outputs y_τ across a number of observations $\tau = 1, \ldots, T$ is minimal.

Thus far, we have given a general description of the state transition and the output equation for open dynamical systems. Without loss of generality we can specify the functions $f()$ and $g()$ by means of a recurrent neural network (RNN)[11, 15]:

$$\begin{array}{ll} \text{state transition} & s_{\tau+1} = \tanh(As_\tau + Bu_\tau) \\ \text{output equation} & y_\tau \quad = Cs_\tau \end{array} \tag{3}$$

Eq. 3 specifies an RNN with weight matrices A, B and C to model the open dynamical system. The RNN is designed as a non-linear state-space model, which is able to approximate any function $f()$ and $g()$. We choose the hyperbolic tangent tanh() as the activation function for the hidden network layer $s_{\tau+1}$. The output equation is specified as a linear function. The RNN output is generated by a superposition of two components: (i) the autonomous dynamics (coded in A), which accumulates information over time (memory), and (ii) the influence of external factors (coded in B).

Note, that the state transition in Eq. 3 does not need an additional matrix leading the hyperbolic tangent tanh() activation function, since the additional matrix can be merged into matrix A. Furthermore, without loss of generality we can use a linear output equation in Eq. 3. If we would have a non-linearity in the output equation (Eq. 3), it could be merged in the state transition equation (Eq. 3). For details see Schäfer et al. [11].

We use the technique of finite unfolding in time[10] to solve the temporal system identification, i. e. for the selection of appropriate matrices A, B and C to minimize the error function (Eq. 1). The underlying idea here is that any RNN can be reformulated to form an equivalent feedforward neural network, if matrices A, B and C are identical in the individual time steps (shared weights). Fig. 1 depicts the resulting RNN.

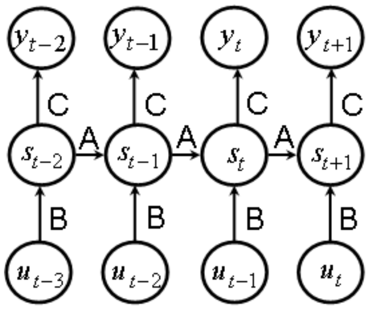


Fig. 1. Basic RNN unfolded in time with shared weights A, B and C

One advantage of the shared matrices is the moderate number of free parameters (weights), which reduces the risk of over-fitting[18]. The actual training is conducted using error backpropagation through time (EBTT) together with a stochastic learning rule[13, 10]. For algorithmic solutions, the reader is referred to the overview article by B. Pearlmutter[9].

Trick 2. Overshooting

In applications we often observed that RNNs tend to focus on only the most recent external inputs in order to explain the dynamics. To balance the information flow, we use a trick called overshooting. Overshooting extends the autonomous system dynamics (coded matrix A) into the future (here $t+2$, $t+3$, $t+4$)[18] (see Fig. 2). In order to describe the development of the dynamics in one of these future time steps adequately, matrix A must be able to transfer information over time. The different instances of matrix A refer to the same prototype matrix A. Thus the shared weights principle allow us to maintain the locality of the correspondence principle (see Trick 1). By this we can compute consistent multi-step forecasts. A corresponding RNN architecture is depicted in Fig. 2. For the RNN we typically use an input preprocessing $u_\tau = x_\tau - x_{\tau-1}$ as the transformation for the raw data x. This avoids trends in the input or target variables of the RNN. The missing external inputs $u_{\tau>0}$ in the future can be interpreted as a constant environment ($u_{\tau>0} \approx 0$). The effectiveness of overshooting depends on the strength of the autonomous dynamics. The stronger the autonomous flow, the better is the forecast accuracy for the future overshooting time steps. Fur-

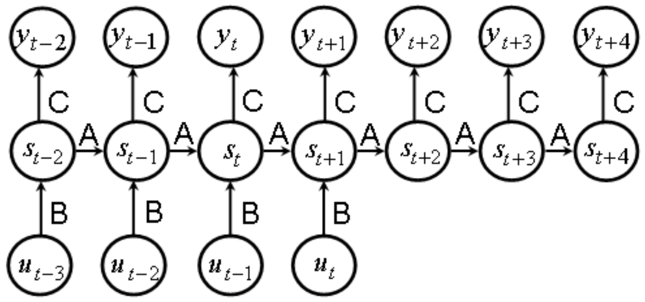


Fig. 2. RNN incorporating overshooting

thermore, overshooting has an implication for the learning itself. Without overshooting, RNNs have the tendency to focus only on short-term input-output relationships. With overshooting the learning has to work out mid- to long-term input-output relationships.

Summarizing, it should be noted, that overshooting generates additional valuable forecast information about the analyzed dynamical system and acts as a regularization method for the learning.

Trick 3. Handling the Uncertainty of the Initial State

One of the difficulties with finite unfolding in time is to find a proper initialization for the first state vector of the RNN. Our next trick takes care of this problem. An obvious solution is to set the first state s^0 equal to *zero*. We assume that the unfolding includes enough (past) time steps such that the misspecification of the initialization phase is overwritten along the state transitions. In other words, the network accumulates information over time and thus, may eliminate the impact of the arbitrary initial state on the network outputs.

Beyond this the modeling can be improved if we are able to make the unfolded RNN less sensitive from the unknown initial state s^0. A first approach to stiff the model against the unknown s^0 is to apply a noise term Θ to the state s^0. A fixed noise term Θ which is drawn from a certain noise distribution is clearly inadequate to handle the uncertainty of the initial state. Since we do not know what is an appropriate level of noise, we have to find a way to estimate the noise level. We propose to apply an adaptive noise Θ, which fits best to the level of uncertainty of the unknown s^0. The characteristic of the adaptive noise term Θ is automatically determined as a by-product of the error backpropagation algorithm.

The basic idea is as follows [8]: We use the residual error ϵ of the neural network as computed by error backpropagation for s^0. The residual error ϵ as measured at the initial state s^0 can be interpreted as the uncertainty which is due to the missing information about the true initial state vector s^0. We disturb the initial state s^0 with a noise term Θ which follows the distribution of the residual error ϵ. Given the uncertain initial states, learning tries to fulfill the output-target relationships along the dynamics. As a result of the learning we get a state transition matrix in form of a contraction, which squeezes out the initial uncertainty. A corresponding network architecture is depicted in Fig. 3.

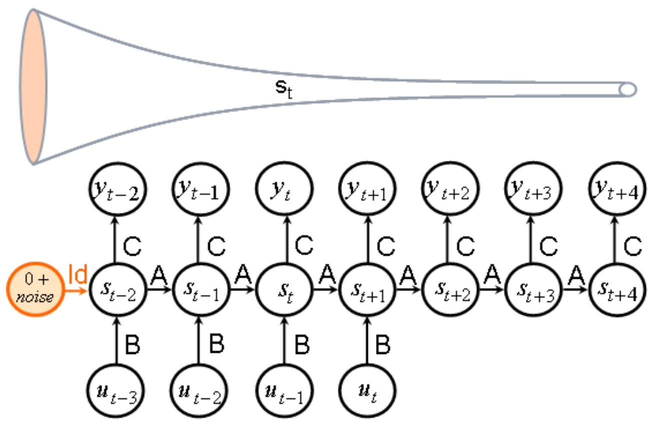


Fig. 3. Handling the uncertainty of the initial state s^0 by applying adaptive noise.

It is important to notice, that the noise term Θ is drawn from the observed residual errors without any assumption on the underlying noise distribution. The desensitization of the network to the initial state vector s^0 can therefore be seen as a self-scaling stabilizer of the modeling.

In general, a time discrete trajectory can be seen as a sequence of points over time. Such a trajectory is comparable to a fine thread in the internal state space. The trajectory is very sensitive to the initial state vector s^0. If we apply noise to s^0, the trajectory becomes a tube in the internal state space. Due to the characteristics of the adaptive noise term, the tube contracts over time. This enforces the identification of a stable dynamical system.

Trick 4. Error Correction Neural Networks (ECNN)

A weakness of the RNN (see Fig. 1 or 2) is, that modeling might be disturbed by unknown external influences or shocks. As a remedy, the next trick called error correction neural networks (ECNN) introduces an additional term $z_\tau = tanh(y_\tau - y_\tau^d)$ in the state transition (4). The term can be interpreted as an correctional factor: The model error $(y_\tau - y_\tau^d)$ at time τ quantifies the misfit and may help to adjust the model output afterwards.

$$\begin{array}{ll} \text{state transition} & s_{\tau+1} = \tanh(As_\tau + Bu_\tau + D\tanh(y_\tau - y_\tau^d)) \\ \text{output equation} & y_\tau \quad = Cs_\tau \end{array} \tag{4}$$

In Eq. 4 the model output y_τ is computed by Cs_τ and compared with the observation y_τ^d. The output clusters of the ECNN which generate error signals during the learning phase are $z_\tau(\tau \leq t)$ and $y_\tau(\tau > t)$. Have in mind, that the target values of the sequence of output clusters z_τ are *zero*, because we want to optimize the compensation mechanism $y_\tau - y^d_\tau$ between the expected value y_τ and its observation y^d_τ. The additional non-linear squashing function in $z_\tau = tanh(y_\tau - y^d_\tau)$ absorbes large errors resp. shocks. A special case occurs at all future time steps $t+1$: here we have no compensation y^d_{t+1} of the internal expected value, and thus the system is offering a forecast $y_{t+1} = Cs_{t+1}$.

The system identification task of (see Eq. 2) is once again solved by finite unfolding in time [3]. Fig. 4 depicts the resulting ECNN[15].

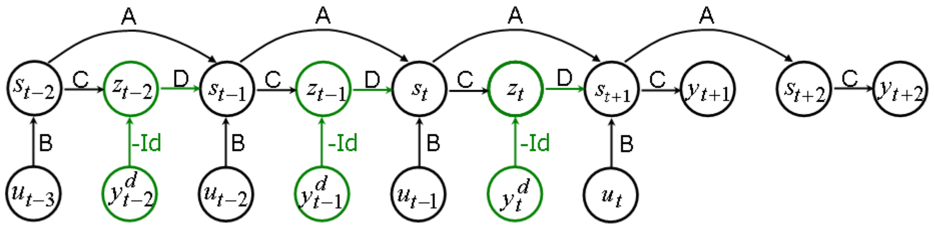


Fig. 4. ECNN incorporating overshooting. Note, that $-Id$ is the fixed negative of an identity matrix, while $z_{t-\tau}$ are output clusters to model the error correction mechanism.

The ECNN has two different inputs: (i) the externals u_τ, which directly influence the state transition, and (ii) the targets y^d_τ. Only the difference between y_τ and y^d_τ has an impact on $s_{\tau+1}$. In the future $\tau > t$, we have no compensation for y_τ and thus compute forecasts $y_\tau = Cs_\tau$. We have successfully applied ECNN models to predict the demand of products and product groups within the context of supply chain management.

Trick 5. Variant-Invariant Separation

For the modeling of high-dimensional dynamical systems, the next trick called variant-invariant separation extends the architecture of the ECNN (Fig. 5) by a coordinate transformation to reduce the dimensionality of the original forecast problem. The dimension reduction is realized in form of a bottleneck sub-network (Fig. 5, left). The compressor E removes time invariant structures from the dynamics. The reconstruction of the complete dynamics (decompression) is done by matrix F. The bottleneck is implicitly connected to the ECNN via the shared matrices E and F [15]. We compress the high-dimensional vector y^d_t to a lower dimensional vector x_t using a bottleneck neural network. The vector x_t contains all relevant information about y^d_t. The ECNN predicts the low dimensional vector x_τ instead of the high dimensional vector y_τ. The error correction compares the expectations $x_\tau = Cs_\tau$ with the transformed observations $-x^d_\tau = E(-y^d_\tau)$. Note, that the negative inputs $-y^d_\tau$ are required by the ECNN to generate the transformed targets $-x^d_\tau$. In our experience we found, that the training of the

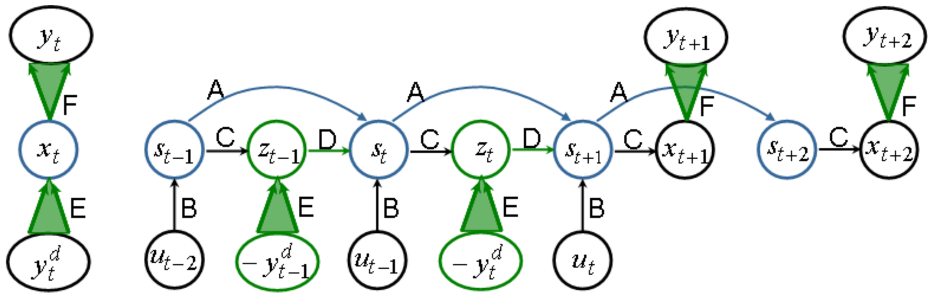


Fig. 5. ECNN with Variant-Invariant Separation

extended ECNN (Fig. 5) is very robust, i. e. the coordinate transformation and the forecasting can be trained in parallel. The time-invariant information is not lost in the bottleneck network, but simply relegated to lower components of variance of the representation. Furthermore, node pruning can be applied to the middle layer. Due to shared weights the result of a pruning in the bottleneck network is transfered to the ECNN branch as well.

We have successfully applied the ECNN depicted in Fig. 5 to forecast electrical load curves and traffic flows. In load forecasting the typical application scenario is that one has to forecast the load curve in 15 minute time buckets, i. e. 96 observations per day. To avoid the error accumulation of a pure iterative model it is more useful to forecast the load curve day-by-day. From our experience a load curve (i. e. 96 observations per day) can be compressed to an $dim \approx 8$ dimensional indicator vector from which the load curve can in turn be reconstructed. From a mathematical viewpoint this approach corresponds to the modeling of dynamical systems on manifolds. More complicated manifolds can be generated by deep bottleneck neural networks.

Trick 6. Historically Consistent Neural Networks (HCNNs)

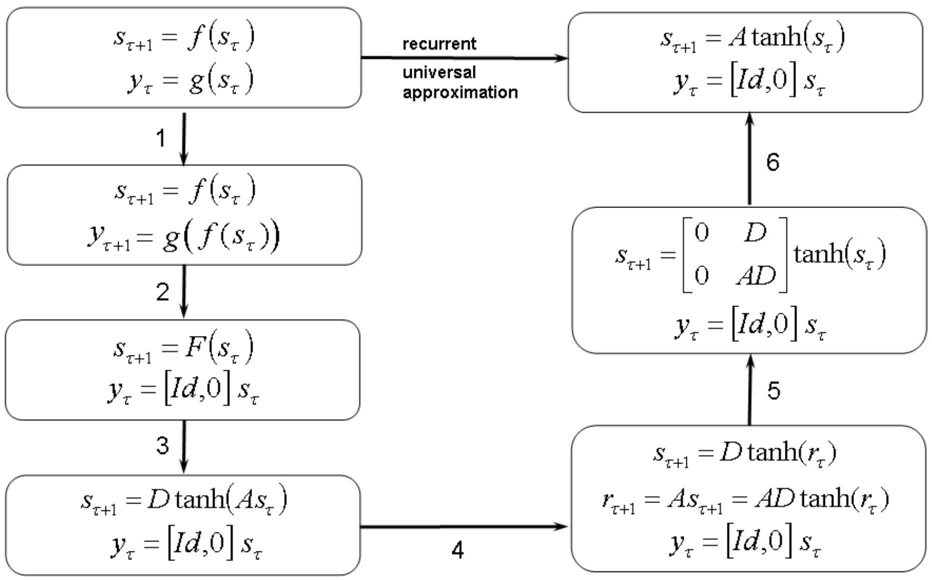
Many real-world technical and economic applications can be seen in the context of large systems in which various (non-linear) dynamics interact with one another (in time). Unfortunately only a small sub-set of variables can be observed. From the sub-set of observed variables we have to reconstruct the hidden variables of the large system in order to understand the dynamics. Here the term *observables* includes the input and output variables in conventional modeling approaches (i. e. $y_\tau := (y_\tau, u_\tau)$). This indicates a consistency problem in the RNN (Fig. 1) or ECNN (Fig. 4) respectively: on the output side, the RNN (ECNN) provides forecasts of the dynamics in the observables y_τ , whereas the input side assumes that the observables y_τ will not change from present time on. This lack of consistency represents a clear contradiction within the model framework. If, on the other hand, we are able to implement a model framework in which common descriptions and forecasts can be used for the trend in all of the observables, we will be in a position to close the open system - in other words, we will model a closed large dynamic system.

The next trick called historically consistent neural networks (HCNNs) introduces a model class which follows the design principles for modeling of large dynamic systems and overcomes the conceptual weaknesses of conventional models. Equation 5 formulates the historically consistent neural network (HCNN).

$$\begin{array}{ll} \text{state transition} & s_{\tau+1} = A\tanh(s_\tau) \\ \text{output equation} & y_\tau \quad = [Id, 0]s_\tau \end{array} \tag{5}$$

The joint dynamics for all observables is characterized in the HCNN (5) by the sequence of states s_τ. The observables ($i = 1, \ldots, N$) are arranged on the first N state neurons s_τ and followed by non-observable (hidden) variables as subsequent neurons. The connector $[Id, 0]$ is a fixed matrix which reads out the observables. The initial state s_0 is described as a bias vector. The bias s_0 and matrix A contain the only free parameters.

Like standard RNNs (Fig. 1) HCNNs also have universal approximation capabilities. The proof for the RNN can be found in [11]. Fig. 6 outlines the proof for HCNNs.



$s_{\tau+1} = f(s_\tau)$, $y_\tau = g(s_\tau)$
recurrent
universal approximation
$s_{\tau+1} = A\tanh(s_\tau)$, $y_\tau = [Id, 0]\, s_\tau$
1
$s_{\tau+1} = f(s_\tau)$, $y_{\tau+1} = g(f(s_\tau))$
2
$s_{\tau+1} = F(s_\tau)$, $y_\tau = [Id, 0]\, s_\tau$
3
$s_{\tau+1} = D\tanh(As_\tau)$, $y_\tau = [Id, 0]\, s_\tau$
4
$s_{\tau+1} = D\tanh(r_\tau)$, $r_{\tau+1} = As_{\tau+1} = AD\tanh(r_\tau)$, $y_\tau = [Id, 0]\, s_\tau$
5
$s_{\tau+1} = \begin{bmatrix} 0 & D \\ 0 & AD \end{bmatrix}\tanh(s_\tau)$, $y_\tau = [Id, 0]\, s_\tau$
6

Fig. 6. Proof of the universal approximation capabilities for HCNNs

As depicted in Fig. 6 the proof of the universal approximation capabilities for HCNN can be divided in six steps:

1. The output equation is shifted one time step into the future and the resulting $s_{\tau+1}$ is substituted by the system transition equation.

2. By combining outputs and state variables into an extended state we get an extended state transition equation. The output of the system is derived from the first components of the extended internal state.
3. For the extended state transition equation we apply the feedforward universal approximation theorem. At least for a finite time horizon this guarantees a small approximation error. Note, that in RNNs at least one large component of the state vector together with the hyperbolic tangent can mimic a bias vector. Thus, we have omitted the explicit notation of a bias vector in the NN equations.
4. In this step we remove one of the two matrices within the state transition equation. We apply a state transformation $r_\tau = As_\tau$. This results in two state transition equations.
5. The two state transition equations can be reorganized in one state transition, which has twice the dimensionality of the original equation.
6. Rewriting the matrix located on front of the tanh activation function results in the claimed formulation for closed systems.

Instead of being applied inside the tanh activation function, matrix A is used outside the tanh activation function. This has the advantage that the states and thus, also the system outputs are not limited to the finite state space $(-1; 1)^n$ created by the tanh(.) nonlinearity. The output equation has a simple and application independent form. Note, that we can only observe the first elements of the state vector.

Fig.7 depicts the HCNN architecture. The HCNN states s_τ are hidden layers

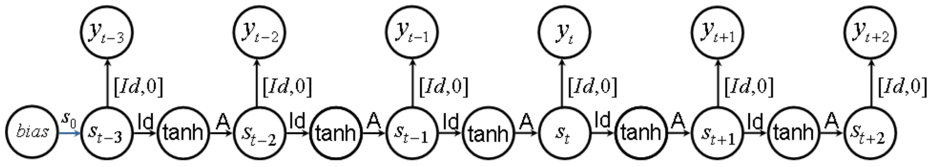


Fig. 7. Historically Consistent Neural Network (HCNN)

with tanh squashing. The forecasts are supplied by the output layers y_τ. There are no target values available for the future time steps. The expected values $y_{\tau>t}$ can be read out at the corresponding future time steps of the network.

Since the HCNN model has no inputs, we have to unfold the neural network along the complete data history. This is different to small recurrent neural networks (see e. g. Fig. 2), where we construct training data patterns in form of sliding windows. The HCNN learns the large dynamics from a single history of observations (i. e. a single training data pattern). Forecasting commodity prices with HCNN, we unfold the neural network over 440 trading days in the past to predict the next 20 days.

Trick 7. Architectural Teacher Forcing (ATF) for HCNNs

In practice we observe that HCNNs are difficult to train since the models do not have any input signals and are unfolded across the complete data set. Our next trick, architectural teacher forcing (ATF) for HCNNs, makes the best possible use of the data from the observables and accelerates the training of the HCNN [20, 14, 9]. The HCNN with integrated teacher forcing is shown in Fig. 8 below. In the HCNN with ATF (Fig. 8) the expected values for all observables up to

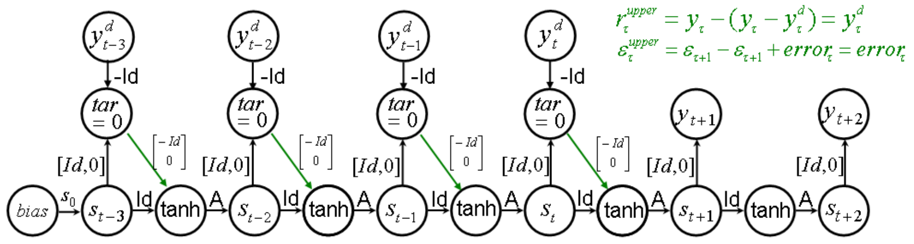

$r_\tau^{upper} = y_\tau - (y_\tau - y_\tau^d) = y_\tau^d$
$\varepsilon_\tau^{upper} = \varepsilon_{\tau+1} - \varepsilon_{\tau+1} + error_\tau = error_\tau$
y_{t-3}^d
y_{t-2}^d
y_{t-1}^d
y_t^d
-Id
tar = 0
$[Id,0]$
$\begin{bmatrix}-Id\\0\end{bmatrix}$
bias
s_0
s_{t-3}
s_{t-2}
s_{t-1}
s_t
s_{t+1}
s_{t+2}
y_{t+1}
y_{t+2}
Id
tanh
A

Fig. 8. HCNN with Architectural Teacher Forcing (ATF)

time $\tau = t$ are replaced with the actual observations. The output layers of the HCNN are given fixed target values of zero. The negative observed values $-y_\tau^d$ for the observables are added to the output layer. This forces the HCNN to create the expected values y_τ to compensate for the negative observed values $-y_\tau^d$. The content of the output layer, i. e. $y_\tau - y_\tau^d$, is now transferred to the first N neurons of the hidden layer r_τ on a component-by-component basis with a minus symbol. In addition, we copy the expected values y_τ from the state s_τ to the intermediate hidden layer r_τ. As a result, the expected values y_τ on the first N components of the state vector s_τ are replaced by the observed values $y_\tau^d = y_\tau - (y_\tau - y_\tau^d)$ (Fig. 8). All connections of the ATF mechanism are fixed. Following the ATF step, the state transition matrix A is applied, to move the system into the next time step. By definition, we have no observations for the observables in future time steps. Here, the system is iterated exclusively upon the expected values. This turns an open into a closed dynamic system. The HCNN in Fig. 8 is equivalent to the architecture in Fig.7, if it converges to zero error in the training. In this case we have solved the original problem.

Trick 8. Sparsity, Dimensionality vs. Connectivity and Memory

HCNNs may have to model ten, twenty or even more observables in parallel over time. It is clear that we have to work with high dimensional dynamical systems (e. g. in our commodity price models we use $dim(s) = 300$). The iteration with a fully connected state transition matrix A of such a dimension is dangerous: Sometimes the matrix vector operations will produce large numbers which will be spread in the recursive computation all over the network and will generate an arithmetic overflow. To avoid this phenomena we can choose a sparse matrix A. Thus, the linear algebra does not accumulate large numbers and the spread of large numbers through the network is damped by the sparsity too.

We have to answer the questions which dimensionality and which sparsity we will choose. In [16] we have worked out that dimensionality and sparsity are related to another pair of meta-parameters: Connectivity (con) and memory length (mem). Connectivity is defined as the number of nonzero elements in each row of matrix A. The memory length is the number of steps from which we have to collect information in order to reach a Markovian state, i. e. the state vector contains all necessary information from the past. We propose the following parameterization for the state dimension ($dim(s)$) and sparsity [16]:

$$\text{Dimension of } A \;\; dim(s) = con \cdot mem \tag{6}$$

$$\text{Sparsity of } A \text{ Sparsity} = random\left(\frac{con}{mem \cdot con}\right) = random\left(\frac{1}{mem}\right) \tag{7}$$

Eq. 7 represents the insight that a sparse system conserves information over a longer time period before it diffuses in the network. For instance a shift register is very sparse and behaves only as a memory, whereas in a fully connected matrix the superposition of information masks the information sources. Let us assume that we have initialized the state transition matrix with a uniformm random sparse matrix A. Following Eq. 7 the more dense parts of A will model the faster sub-dynamics within the overall dynamics, while the highly sparse parts of A will focus on slow subsystems. As a result a sparse random initialization allows the combined modeling of systems on different time scales.

Unfortunately, Eq. 6 favors very large dimensions. Our earlier work on the subject (see [16]) started with the predefinition of the systems memory length mem, because for RNNs the memory length is equal to the length of the past unfolding in time. On the other hand, connectivity has to be chosen larger than the number of the observables. Working with HCNNs the memory length is less important, because we unfold the neural network along the whole data horizon. Here the connectivity plays the superior role. From our experience we know that the EBTT algorithm works stably with a connectivity which is equal or smaller than 50 ($con \leq 50$). For computational performance we usually limit the state dimensionality to $dim(s) = 300$. This implies a sparsity of $50/300 \approx 17\%$. We leave the fine tuning of the parameters to the EBTT learning.

We propose to initialize the neural network with a randomly chosen sparsity grid. The sparsity grid is therefore chosen arbitrary and not optimized by e. g. pruning algorithms. This raises the question if a random sparse initialization biases the network towards inferior solutions. This is handled by ensemble forecasts. We have performed ensemble experiments with different sparsity grids versus ensembles based on the same sparsity grid. We found, that the average of the ensemble as well as the ensemble width are unaffected by the initialization of the sparsity grid (for more details on ensemble forecasting see Trick 12). These considerations hold only for large systems.

Trick 9. Causal-Retro-Causal Neural Networks (CRCNNs)

The fundamental idea of the HCNN is to explain the joint dynamics of the observables in a causal manner, i. e. with an information flow from the past to the future. However, rational planning is not only a consequence of a causal information flow but also of anticipating future developments and responding to them on the basis of a certain goal function. This is similar to the adjoint equation in optimal control theory [6]. In other words a retro causal information flow is equivalent to asking for the motivation of a behavior as a goal. In turn, this is the anchor point for the reconstruction of the dynamics.

In order to incorporate the effects of rational decision making and planning into the modeling, the next trick introduces causal-retro-causal neural networks (CRCNNs). The idea behind the CRCNN is to enrich the causal information flow within the HCNN, which is directed from the past to the future, by a retro-causal information flow, directed from the future into the past. The CRCNN model is given by the following set of equations 8.

$$\begin{array}{ll} \text{causal state transition} & s_\tau = A\tanh(s_{\tau-1}) \\ \text{retro-causal state transition} & s'_\tau = A'\tanh(s'_{\tau+1}) \\ \text{output equation} & y_\tau = [Id,0]s_\tau + [Id,0]s'_\tau . \end{array} \tag{8}$$

The output equation y_τ of the CRCNN (Eq. 8) is a mixture of causal and retro-causal influences. The dynamics of all observables is hence explained by a sequence of causal (s_τ) and retro-causal states s'_τ using transition matrices A and A' for the causal and retro-causal information flow. Upon the basis of Eq. 8, we draw the network architecture for the CRCNN as depicted in Fig. 9.

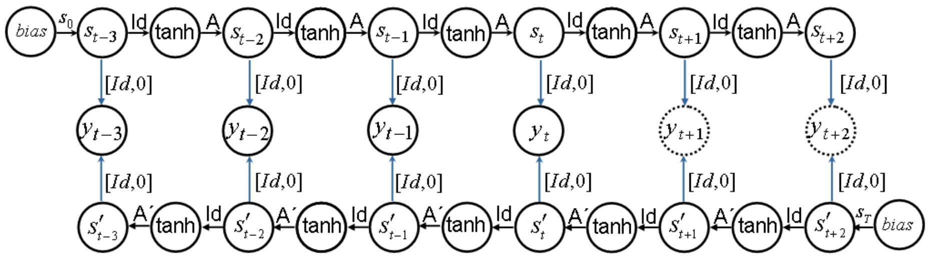
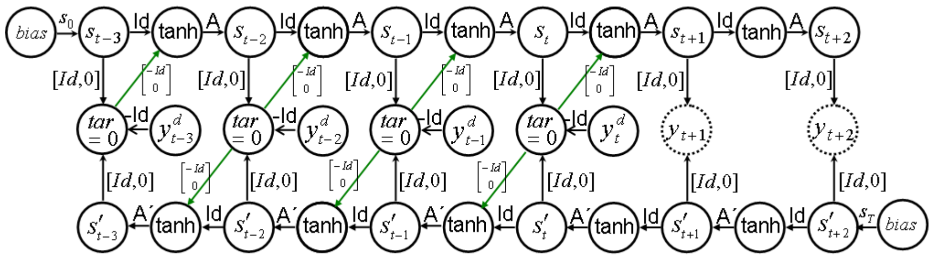


Fig. 9. A Causal-Retro-Causal Historically Consistent Neural Network (CRCNN)

Trick 10. Architectural Teacher Forcing (ATF) for CRCNNs

The CRCNN (Fig. 9) is also unfolded across the entire time path, i. e. we learn the unique history of the system. Likewise to the training of the HCNN, CRCNNs are difficult to train. Our next trick called architectural teacher forcing (ATF) for CRCNNs formulates TF as a part of the CRCNN architecture, which allows us to learn the CRCNN using the standard EBTT algorithm[3]. ATF enables us to exploit the information contained in the data more efficiently and accelerates the training itself. Fig. 10 depicts the CRCNN architecture incorporating ATF.



$$0 = \Delta_t + \Delta'_t - y^d_t \quad \Rightarrow \quad \Delta'_t = y^d_t - \Delta_t = \Delta'_t - (\Delta_t + \Delta'_t - y^d_t)$$

$$0 = \Delta_t + \Delta'_t - y^d_t \quad \Rightarrow \quad \Delta_t = y^d_t - \Delta'_t = \Delta_t - (\Delta_t + \Delta'_t - y^d_t)$$

Fig. 10. Extended CRCNN with an architectural Teacher Forcing (ATF) mechanism

Let us explain the ATF mechanism in the extended CRCNN model (Fig. 10): The extended CRCNN uses a causal-retro-causal network to correct the error of the opposite part of the network in a symmetric manner. In every time step $\tau \leq t$ the expected values y_τ are replaced by the observations y^d_τ using the intermediate tanh() layers for the causal and for the retro-causal part. Since the causal and the retro-causal part *jointly* explain the observables y_τ, we have to inject the causal into the retro-causal part and vice versa. This is done by compensating the actual observations $-y^d_\tau$ with the output of the causal $(\varDelta_\tau)$ *and* the retro-causal part $(\varDelta'_\tau)$ within output layers with fixed target values of zero. The resulting content $(\varDelta_\tau + \varDelta'_\tau - y^d_\tau = 0)$ of the output layers is negated and transferred to the causal and retro-causal part of the CRCNN using the fixed $[-Id, 0]'$ connector. Within the intermediate tanh() layers the expectations of y_τ are replaced with the actual observations y^d_τ, whereby the contribution to y_τ from the opposite part of the CRCNN is considered. Note, that ATF does not lead to a larger number of free network parameters, since all new connections are fixed and are used only to transfer data in the NN. In future direction $\tau > t$ the CRCNN is iterated exclusively on the basis of expected values.

The usage of ATF does not reintroduce an input / output modeling, since we replace the expected value of the observables with their actual observations, while simultaneously considering the causal and retro-causal part of the dynamics. For sufficiently large CRCNNs and convergence of the output error to zero, the architecture in Fig. 10 converges towards the fundamental CRCNN architecture shown in Fig. 9. The advantage of the CRCNN is, that it allows a fully dynamical superposition of the causal and retro-causal information flows.

The CRCNN depicted in Fig. 10 describes a dynamics on a manifold. In every time step the information flows incorporates closed loops, which technically can be seen as equality constraints. These contraints are only implicitly defined through the interaction of the causal and retro-causal parts. The closed loops in the CRCNN architecture (Fig. 10) lead to fix-point recurrent substructures within the model, which are hard to handle with EBTT. As a remedy we propose a solution concept similar to Trick 7: We embedd the CRCNN model into a larger network architecture, which is easier to solve and converges to the same solution as the original system. Fig. 11 depicts an initial draft for such an embedding.

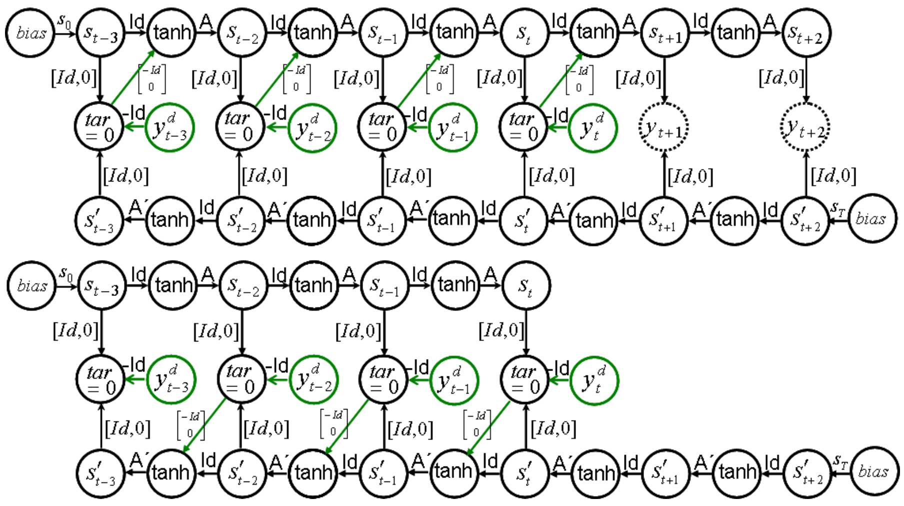


Fig. 11. Asymmetric split of ATF in CRC neural networks

The extended architecture in Fig. 11 is a duplication of the origonal model depicted in Fig. 10. The CRCNN architecture in Fig. 11 does not contain closed loops, because we splitted the ATF mechanism for the causal and retro-causal part into two branches. It is important to notices that these branches are implicitly connected through the shared weights in the causal and retro-causal part. If this architecture converges, the ATF is no longer required and we have two identical copies of the CRCNN model depicted in Fig. 9. The solution proposed for the embedding is not the only feasible way to handle the fix-point loops. We will outline alternative solutions in an upcomming paper. The CRCNN is the basis for our projects on forecasting commodity prices.

Trick 11. Stable & Instable Information Flows in Dynamical Systems

Following Trick 3 it is natural to apply noise in the causal as well as in retro-causal branch (see also Fig. 12). This should improve the stability of both time directions resp. information flows. In CRCNNs this feature has a special interpretation: Stability in the causal information flow means that the uncertainty in the beginning is damped along the time path from past to future. Instabilty in the causal system means that a small disturbance in the past diffuses to very different future scenarios under a chaotic regime (see Fig. 12, upper part).

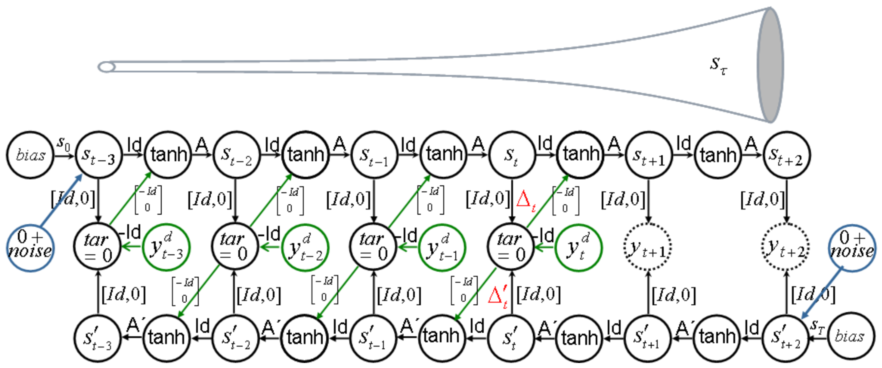


Fig. 12. An instable causal dynamics is converted to a stable retro-causal dynamics.

If the causal information flow of a sub-dynamics is instable, then the retro-causal decription of this sub-system is stable. On the other hand, an instable retro-causal dynamics is stable from a causal perspective. In a combined causal-retro-causal neural network the causal *and* the retro-causal branch can be simultaneously stable, even if the underlying dynamics is partially instable. In order to enforce the stability of the causal and the retro-causal part we apply noise at the orgins of both branches.[1]

Trick 12. Uncertainty and Risk

The experience gained during the latest financial crisis has triggered a far-reaching discussion on the limitations of quantitative forecasting models and made investors very conscious of risk[2]. In order to understand risk distributions, traditional risk management uses diffusion models. Risk is understood as a random walk, in which the diffusion process is calibrated by the observed past error of the underlying model[7]. In contrast the next trick called uncertainty and risk focuses on ensemble forecasts in order to provide important insights into complex risk relationships, since internal model (unobserved) variables can be reconstructed from the trend in observed variables (observables).

If the system identification is calculated repeatedly for HCNNs or CRCNNs, an ensemble of solutions will be produced, which all have a forecast error of zero in the past, but which differ from one another in the future. Since every HCNN or CRCNN model gives a perfect description of the observed data, the complete ensemble is the true solution. A way to simplify the forecast is to take

[1] Thanks to Prof. Jürgen Jost, MPI Leipzig, for a fruitful discussion on this topic.

the arithmetical average of the individual ensemble members as the expected value, provided the ensemble histogram is unimodal in every time step.

In addition to the expected value, we consider the bandwidth of the ensemble, i. e. its distribution. The form of the ensemble is governed by differences in the reconstruction of the hidden system variables from the observables: for every finite volume of observations there is an infinite number of explanation models which describe the data perfectly, but differ in their forecasts, since the observations make it possible to reconstruct the hidden variables in different forms during the training. In other words, our risk concept is based on the partial observability of the world, leading to different reconstructions of the hidden variables and thus, different future scenarios. Since all scenarios are perfectly consistent with the history, we do not know which of the scenarios describes the future trend best and risk emerges.

This approach directly addresses the model risk. For HCNN and CRCNN modeling we claim that the model risk is equal to the forecast risk. The reasons can be summarized as follows: First, HCNNs are universal approximators, which are therefore able to describe every market scenario. Second, the form of the ensemble distribution is caused by an underlying dynamics, which interpret the market dynamics as the result of interacting decisions[17]. Third, in experiments we have shown that the ensemble distribution is independent from the details of the model configuration, if we use large models and large ensembles.

Let us exemplify our risk concept. The diagram below (Fig. 13, left) shows the approach applied to the Dow Jones Industrial Index (DJX). For the ensemble, a HCNN was used to generate 250 individual forecasts for the DJX. For

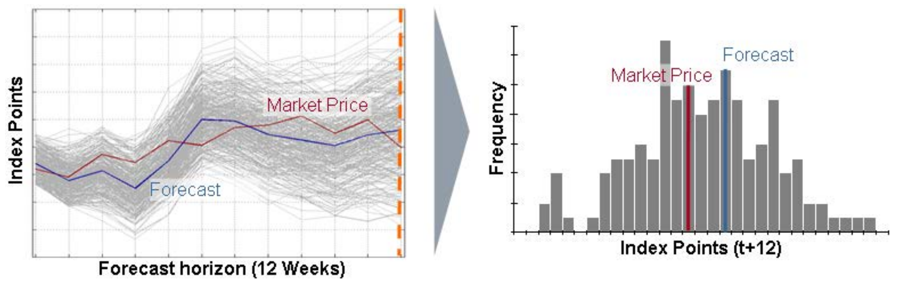


Fig. 13. HCNN ensemble forecast for the Dow Jones Index (12 weeks forecast horizon), left, and associated index point distribution for the ensemble in time step $t + 12$, right.

every forecast date, all of the individual forecasts for the ensemble represent the empirical density function, i. e. a probability distribution over many possible market prices at a single point in time (see Fig. 13, right). It is noticeable that the actual development of the DJX is always within the ensemble channel (see gray lines, Fig. 13, left). The expected value for the forecast distribution is also an adequate point forecast for the DJX (see Fig. 13, right).

3 Conclusion and Outlook

Recurrent neural networks model dynamical systems in the form of non-linear state space models. Just like any other NN, the equation of the RNNs, ECNNs, HCNNs or CRCNNs can be expressed as an architecture which represents the individual layers in the form of nodes and the connections between the layers in the form of links. In the graphical architecture we can apply local learning algorithms like error back propagation and an appropriate (stochastic) learning rule to train the NN[13, 17, 3]. This relationship is called the correspondence principle between equations, architectures and the local algorithms associated with them (**Trick 1**).

Finite unfolding in time of RNN using shared weight matrices enables us to stick to the correspondence principle. Overshooting enforces the autonomous dynamics and enables long-term forecasting (**Trick 2**), whereas an adaptive noise term handles the uncertainty of the finite unfolding in time (**Trick 3**).

ECNN utilizes the previous model error as an additional input. Hence, the learning can interpret the models misfit as an external shock which is used to guide the model dynamics afterwards. This allows us to prevent the autonomous part of the model to adapt misleading inter-temporal causalities. If we know that a dynamical system is influenced by external shocks, the error correction mechanism of the ECNN is an important prestructuring element of the networks architecture to compensate missing inputs (**Trick 4**).

Extending the ECNN by variants-invariants separation, one is able to include additional prior structural knowledge of the underlying dynamics into the model. The separation of variants and invariants with a bottleneck coordinate transformation allows to handle high dimensional problems (**Trick 5**).

HCNNs model not just an individual dynamics, but complex systems made up of a number of interacting sub-dynamics. HCNNs are symmetrical in their input and output variables, i. e. the system description does not draw any distinction between input, output and internal state variables. Thus, an open system becomes a closed system (**Trick 6**). Sparse transition matrices enable us to model different time scales and stabilize the training (**Trick 8**). Causal and retro-causal information flow within an integrated model (CRCNN) can be used to model rational planning and decision making in markets. CRCNNs dynamically combine causal and retro-causal information to describe the prevailing market regime (**Trick 9**). Architectural teacher forcing can be applied to efficiently train the HCNN or CRCNN(**Trick 7 and 10**). An architectural extension (see Fig. 12) enables us to balance the causal and retro-causal information flow during the learning of the CRCNN (**Trick 11**).

We usually work with ensembles of HCNN or CRCNN to predict commodity prices. All solutions have a model error of zero in the past, but show a different behavior in the future. The reason for this lies in different ways of reconstructing the hidden variables from the observations and is independent of different random sparse initializations. Since every model gives a perfect description of the observed data, we can use the simple average of the individual forecasts as the expected value, assuming that the distribution of the ensemble is unimodal.

The analysis of the ensemble spread opens up new perspectives on market risks. We claim that the model risk of a CRCNN or HCNN is equal to the forecast risk (**Trick 12**).

Work currently in progress aims to improve the embedding of the CRCNN architecture (see Fig. 10) in order to simplify and stabilize the training. On the other hand, we analyze the micro-structure of the ensembles and implement the models in practical risk management and financial market applications.

All NN architectures and algorithms are implemented in the Simulation Environment for Neural Networks (SENN), a product of Siemens Corporate Technology. Work is partially funded by German Federal Research Ministry (BMBF grant Alice, 01 IB10003 A-C).

References

1. Calvert D. and Kremer St.: Networks with Adaptive State Transitions, in: Kolen J. F. and Kremer, St. (Ed.): A Field Guide to Dynamical Recurrent Networks, IEEE, 2001, pp. 15-25.
2. Föllmer, H.: Alles richtig und trotzdem falsch?, Anmerkungen zur Finanzkrise und Finanzmathematik, in: MDMV 17/2009, pp. 148-154
3. Haykin S.: Neural Networks and Learning Machines, 3rd Edition, Prentice Hall, 2008.
4. Hull, J.: Options, Futures & Other Derivative Securities. Prentice Hall, 2001.
5. Hornik, K., Stinchcombe, M. and White, H.: Multilayer Feedforward Networks are Universal Approximators, Neural Networks, 2, 1989, pp. 359-366.
6. Kamien, M. and Schwartz, N.: Dynamic Optimization: The Calculus of Variations and Optimal Control in Economics and Management, Elsevier Science, 2nd edition, October 1991.
7. McNeil, A., Frey, R. and Embrechts, P.: Quantitative Risk Management: Concepts, Techniques and Tools, Princeton University Press, Princeton, New Jersey, 2005.
8. Neuneier R. and Zimmermann H. G.: *How to Train Neural Networks*, in: *Neural Networks: Tricks of the Trade*, Springer, Berlin, 1998, pp. 373-423.
9. Pearlmutter B.: Gradient Calculations for Dynamic Recurrent Neural Networks, in: A Field Guide to Dynamical Recurrent Networks, Kolen, J.F.; Kremer, St. (Ed.); IEEE Press, 2001, pp. 179-206.
10. Rumelhart D. E., Hinton G. E. and Williams R. J.: Learning Internal Representations by Error Propagation, in: Rumelhart D. E., McClelland J. L., et al. (Ed.): Parallel Distributed Processing, Vol. 1: Foundations, M.I.T. Press, Cambridge, 1986.
11. Schäfer, A. M. and Zimmermann, H.-G.: Recurrent Neural Networks Are Universal Approximators. ICANN, Vol. 1., 2006, pp. 632-640.
12. Wei W. S.: Time Series Analysis: Univariate and Multivariate Methods, Addison-Wesley Publishing Company, N.Y., 1990.
13. Werbos P. J.: Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, PhD Thesis, Harvard University, 1974.
14. Williams R. J. and Zipser, D.: A Learning Algorithm for continually running fully recurrent neural networks, Neural Computation, Vol. 1, No. 2, 1989, pp. 270-280.
15. Zimmermann, H. G., Grothmann, R. and Neuneier, R.: Modeling of Dynamical Systems by Error Correction Neural Networks. In: Soofi, A. und Cao, L. (Ed.): Modeling and Forecasting Financial Data, Techniques of Nonlinear Dynamics, Kluwer, 2002.

16. Zimmermann, H. G., Grothmann, R., Schäfer, A. M. and Tietz, Ch.: Modeling Large Dynamical Systems with Dynamical Consistent Neural Networks, in New Directions in Statistical Signal Processing. Haykin, S. et al (Ed.), MIT Press, Cambridge, Mass., 2006.
17. Zimmermann, H. G.: Neuronale Netze als Entscheidungskalkül. In: Rehkugler, H. und Zimmermann, H. G. (Ed.): Neuronale Netze in der Ökonomie, Grundlagen und wissenschaftliche Anwendungen, Vahlen, Munich 1994.
18. Zimmermann H. G. and Neuneier R.: Neural Network Architectures for the Modeling of Dynamical Systems, in: A Field Guide to Dynamical Recurrent Networks, Kolen, J. F.; Kremer, St. (Ed.); IEEE Press, 2001, pp. 311-350.
19. Zimmermann H. G., Grothmann R., Tietz Ch. and v. Jouanne-Diedrich H.: Market Modeling, Forecasting and Risk Analysis with Historical Consistent Neural Networks, in: Hu, B. et al. (Eds.): Operations Research Proceedings 2010, Selected Papers of the Annual Int. Conferences of the German OR Society (GOR), Munich, Sept. 2010, Berlin and Heidelberg, Springer 2011.
20. Zimmermann, H. G., Grothmann, R. and Tietz, Ch.: Forecasting Market Prices with Causal-Retro-Causal Neural Networks, in: Klatte, Diethard; Lẗhi, Hans-Jakob; Schmedders, Karl (Eds.): Operations Research Proceedings 2011, Selected Papers of the Int. Conference on Operations Research 2011 (OR 2011), Zurich, Switzerland, Springer 2012.