SIEMENS

Corporate Technology

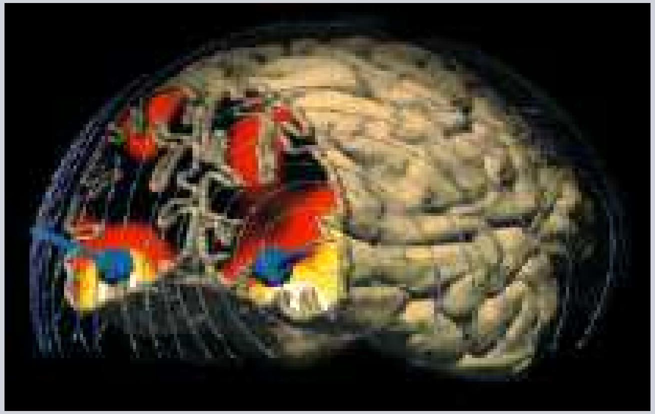
# A Short Course in Neural Networks

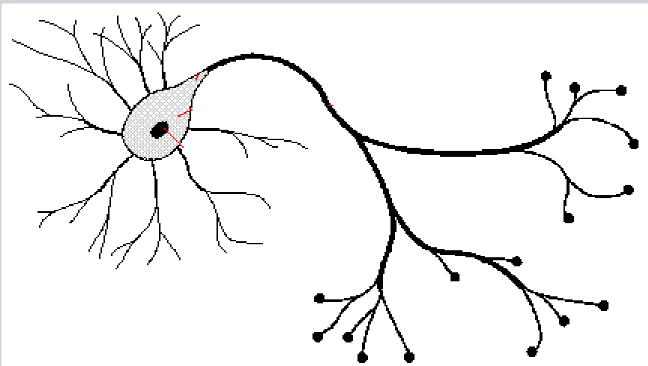
Ralph Grothmann, PhD

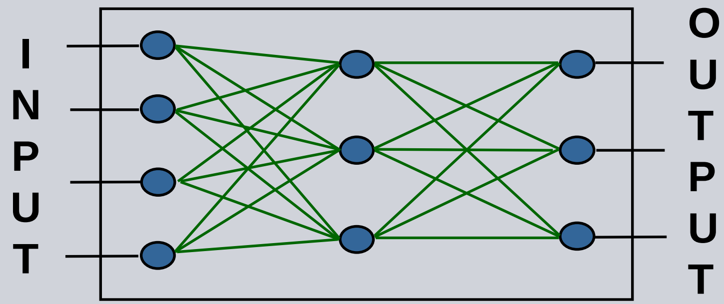
Ralph.Grothmann@siemens.com

# Neural Networks - from Biology to Mathematics

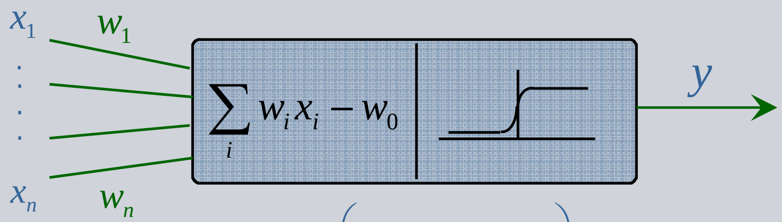
From the modeling of biology to the learning of high dimensional, nonlinear systems








$$y = f\left(W_2\, f\left(W_1 x - w_1\right) - w_2\right)$$

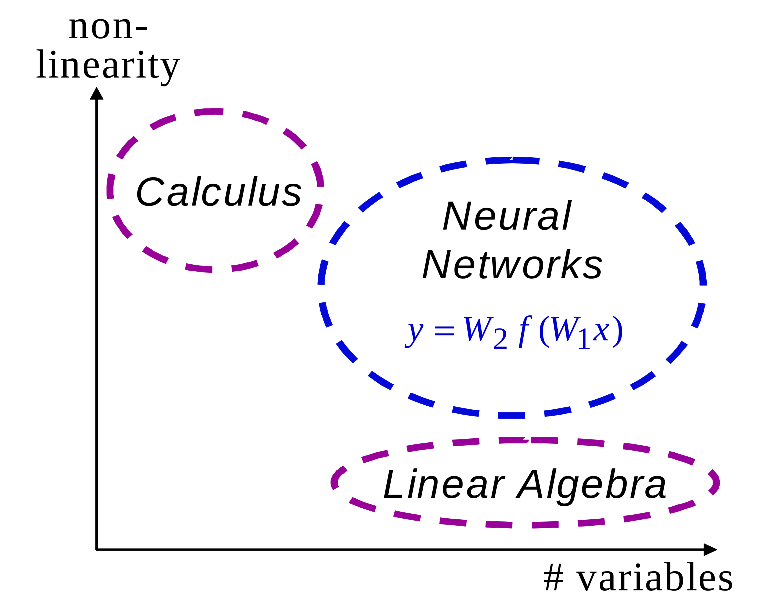



$$y = f\left(\sum_i w_i\, x_i - w_0\right)$$

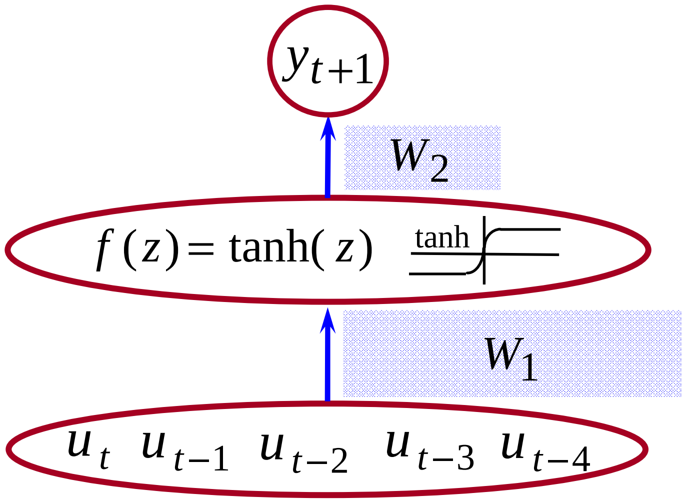
Distinct to linear superpositions of basis functions, NN are *composed substructures*

# Forecasting with Feedforward Neural Networks




Dependent on historical data we search for a function h(…) modeling the shift to the future.




$$y_{t+1} = h(u_t, u_{t-1}, \ldots) = W_2 f(W_1 x)$$

$$\sum_{t=1}^{T} \left(y_t - y_t^d\right)^2 \to \min_h$$

**Existence Theorem:**

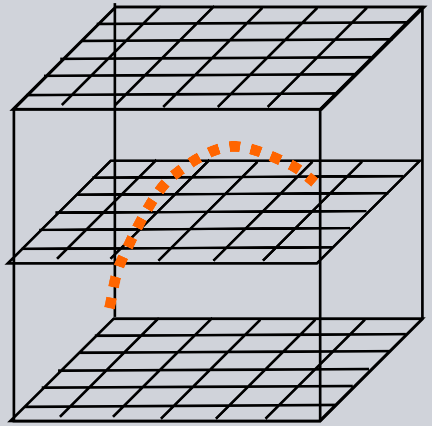
(Hornik, Stinchcombe, White 1989)

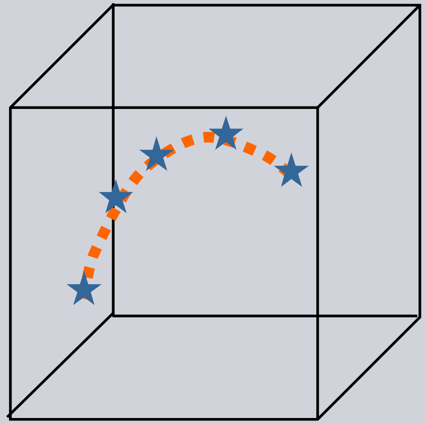
A 3-layer network with sufficient hidden neurons can approximate any continuous function on a compact domain.

SIEMENS

# The Curse of Dimensionality in Approximation Theory





## The curse of dimensionality in Standard Approximation:

$$f(x) \approx \sum_{j=1}^{m} v_j \, b_j(x) \quad \text{with} \quad \left\|\{v_j\}\right\| \approx c^{\dim(x)}$$

This is a linear superposition of basis functions – their number & the number of parameters increase exponentially with dim(x).

## Neural Networks escape the curse of dimensionality:

$$f(x) \approx \sum_{j=1}^{m} v_j \, b(w_j, x) \quad \text{with} \quad \left\|\{v_j, w_j\}\right\| \approx Var(f)$$

The independence of the number of parameters from the input dimension is paid with nonlinear optimization.

## Support Vector Machines offer an alternative remedy:

$$f(x) \approx \sum_{j=1}^{m} v_j \, b(x - x_j) \quad \text{with} \quad \left\|\{v_j\}\right\| \approx \left\|data\right\| \& Var(f)$$

Here we have a linear superposition of basis functions, which are chosen as part of the data - which can be a drawback.

SIEMENS

# Error Backpropagation - Correspondence between Architecture & Algorithm

target

$out_2 = f_2(netin_2)$ $\qquad dev_2 = out_2 - target$

Output

$netin_2 = W_2 out_1$ $\qquad \partial_2 = f_2'(netin_2) dev_2$

$\boxed{\frac{\partial E_t}{\partial W_2} = \partial_2 out_1^T}$

$out_1 = f_1(netin_1)$ $\qquad dev_1 = W_2^T \partial_2$

Hidden

$netin_1 = W_1 out_0$ $\qquad \partial_1 = f_1'(netin_1) dev_1$

$\boxed{\frac{\partial E_t}{\partial W_1} = \partial_1 out_0^T}$

$out_0 = netin_0$ $\qquad dev_0 = W_1^T \partial_1$

Input

$netin_0 = input$ $\qquad \partial_0 = Id \cdot dev_0 = \frac{\partial E_t}{\partial x}$

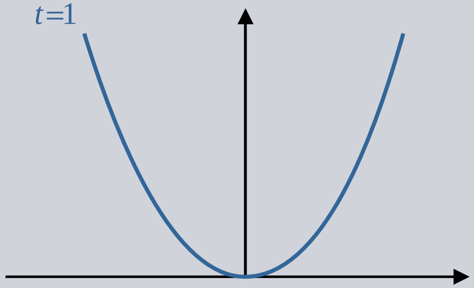
$$E = \frac{1}{T}\sum_{t=1}^{T} E_t = \frac{1}{T}\sum_{t=1}^{T} (y_t - y_t^d)^2$$

$$y = f_2(W_2\ f_1(W_1 x))$$

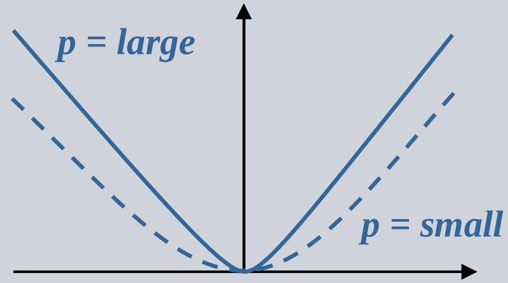
- By the forward & backward flows, $\frac{\partial E_t}{\partial W_1}, \frac{\partial E_t}{\partial W_2}$ are efficiently computed.
- Because of the local algorithm, we can easily extend the network.
- In case of $f(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ we get
  $f'(netin) = 1 - (f(netin))^2 = 1 - out^2$
- In case of $f(z) = logistic(z) = \frac{1}{1+e^{-z}}$ we get
  $f'(netin) = f(netin)(1 - f(netin)) = out(1 - out)$

# Outlier Handling on Targets - Robust Error Functions & Derivatives

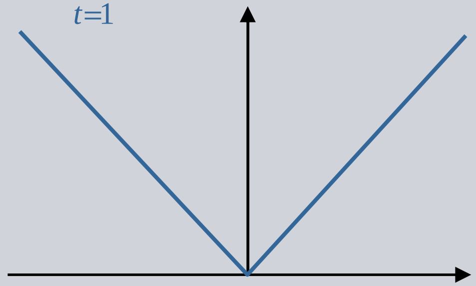
$$\frac{1}{T}\sum_{t=1}^{T}\frac{1}{2}(y_t - y_t^d)^2 \rightarrow \min$$



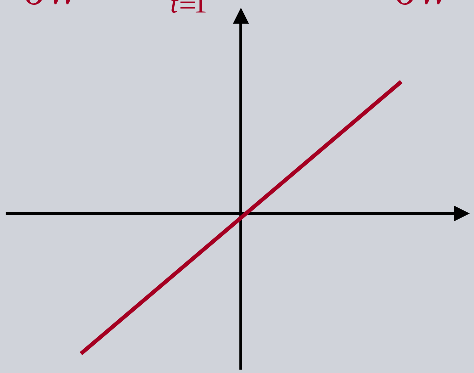
$$\frac{1}{T}\sum_{t=1}^{T}\frac{1}{p}\ln\cosh(p(y_t - y_t^d)) \rightarrow \min$$



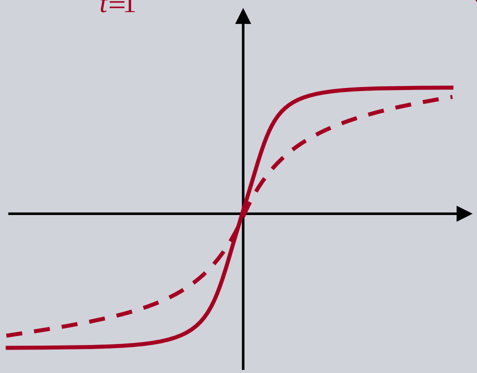

$$\frac{1}{T}\sum_{t=1}^{T}\left\|y_t - y_t^d\right\| \rightarrow \min$$



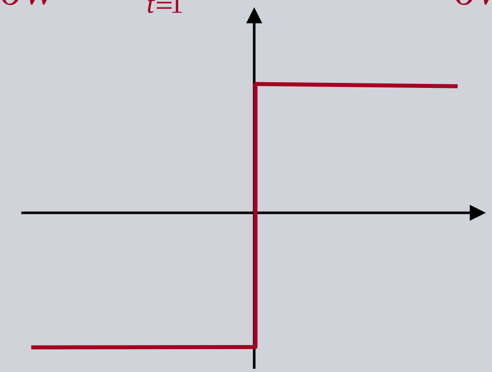
$$\frac{\partial E}{\partial w} = \frac{1}{T}\sum_{t=1}^{T}(y_t - y_t^d)\frac{\partial y_t}{\partial w}$$



$$\frac{\partial E}{\partial w} = \frac{1}{T}\sum_{t=1}^{T}\tanh(p(y_t - y_t^d))\frac{\partial y_t}{\partial w}$$



$$\frac{\partial E}{\partial w} = \frac{1}{T}\sum_{t=1}^{T}sign(y_t - y_t^d)\frac{\partial y_t}{\partial w}$$

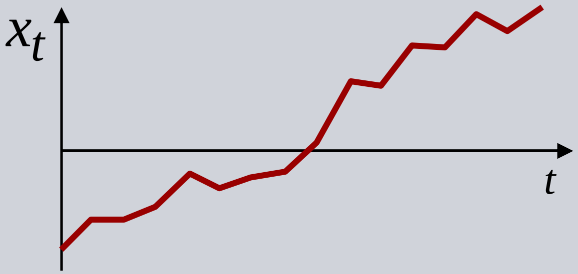


- easy to use derivative
- large impact of outliers

- easy to use derivative
- no impact of outliers (robust)

- complicated derivative
- no impact of outliers (robust)

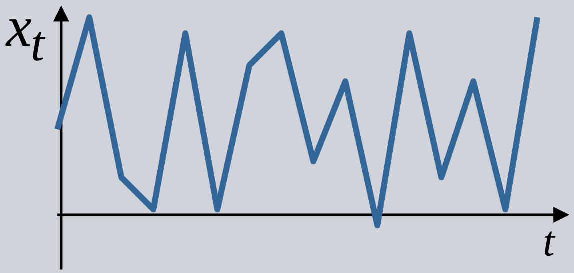
# Proposals for External and Internal Data Preprocessing



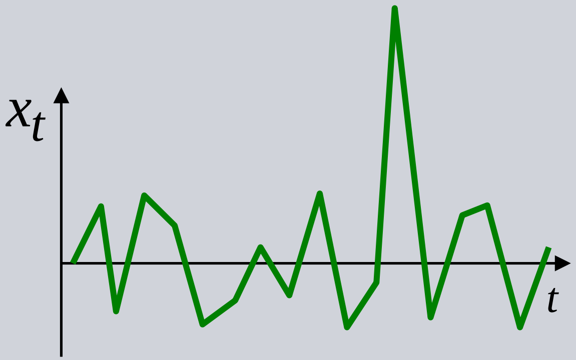

$$\tilde{x}_t = x_t - x_{t-n}$$

n = forecast horizon (smoothing)

$$\tilde{x}_t = \ln\left(\frac{x_t}{x_{t-n}}\right) \approx \frac{x_t - x_{t-n}}{x_{t-n}}$$

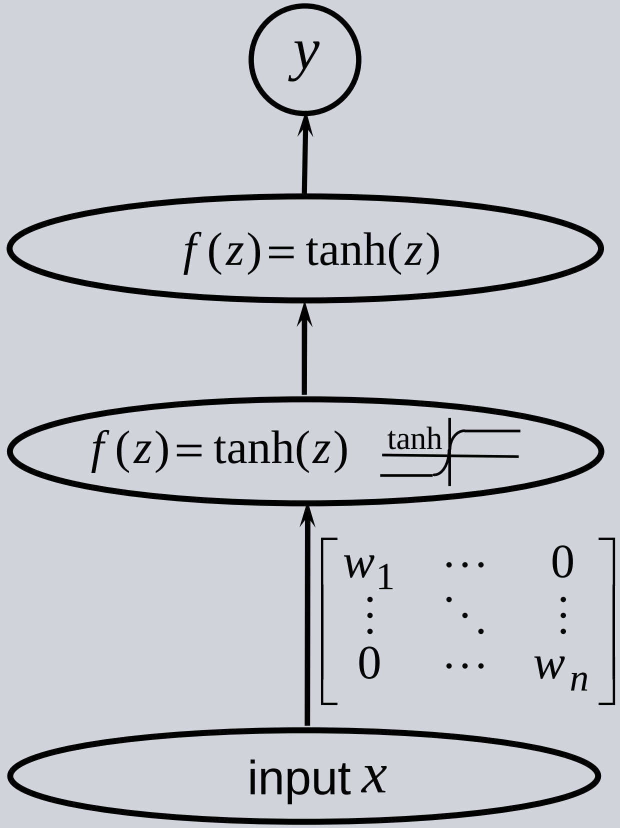



$$\tilde{x}_t = scale(x_t) = \frac{x_t - \bar{x}}{\sqrt{\Sigma(x_t - \bar{x})^2}}$$



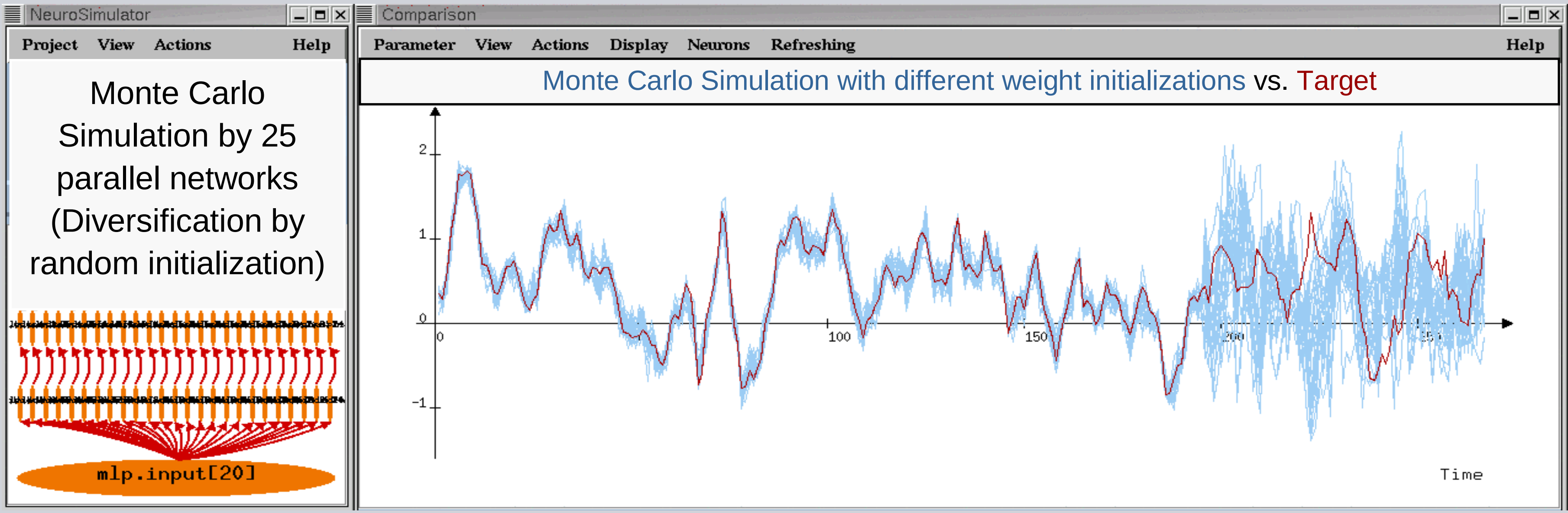

$$\tilde{x}_t = \frac{x_t - median(x)}{median(|x_t - median(x)|)}$$

The unknown level of outlier damping is solved in form of a parameter identification task:



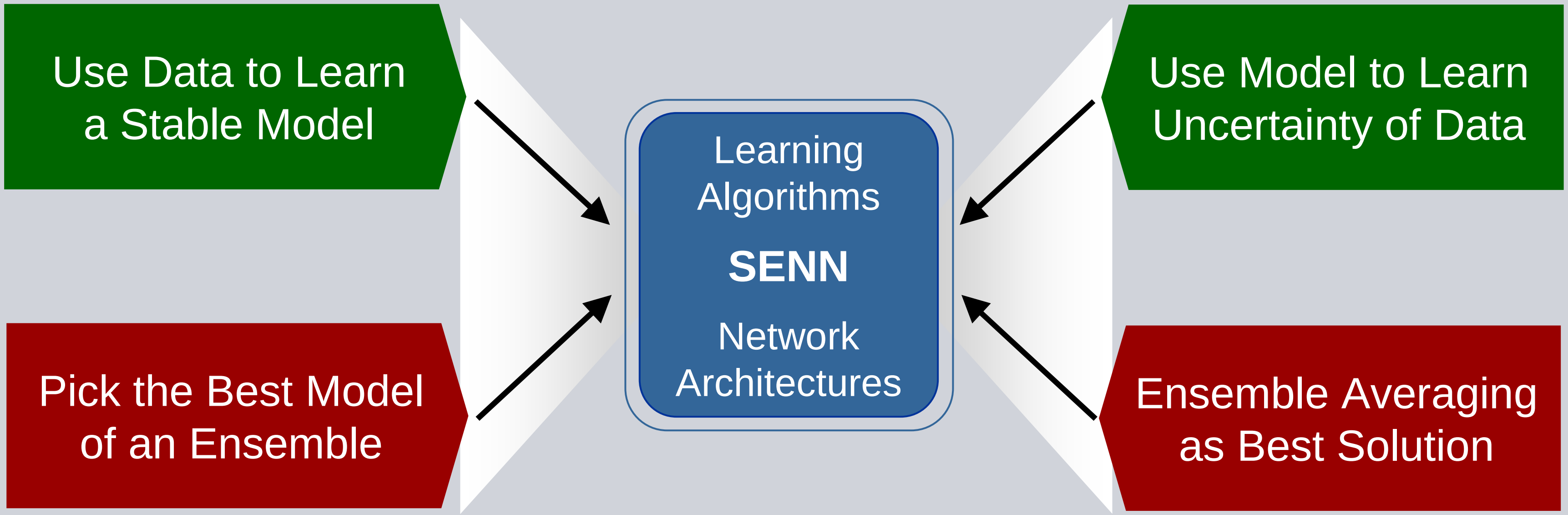
# Uncertainty in Model Building

- In nonlinear modeling, different weight initializations may end up in different local minima. This differentiation shows up on the training as well as on the test set.
- Large networks may be underdetermined. The random substructures do not cause problems on the training set but cause a differentiation on the test set.
- The over-parameterization is even helpful to lower the local minima problem, but we pay the price by the uncertainty of the generalization behavior.



# Techniques for Nonlinear System Identification

Backpropagation allows an efficient computation of gradients, but how to do the weight update to get a stable model? Can we use the model to evaluate the data quality & filter corrupted data.




Given an ensemble of models, Occams razor defines the best model as the most parsimonious. Bayesian analysis defines the best solution as the average solution of the model ensemble.

SIEMENS

## Learning Structure from Data - Learning Rules for Stochastic Search

Task: $E = \frac{1}{T}\sum_{t=1}^{T} E_t = \frac{1}{T}\sum_{t=1}^{T}\left(NN(x_t, w) - y_t^d\right)^2 \rightarrow \min_w$

Notation: $g_t = \frac{\partial E_t}{\partial w}$, $g = \frac{1}{T}\sum_{t=1}^{T} g_t$

Steepest descent learning: $\Delta w = \eta \cdot (-g)$ = step length · search direction

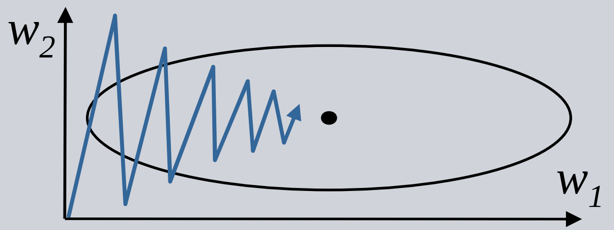
$$E(w + \Delta w) = E(w) + g^T \Delta w + \frac{1}{2}\Delta w^T G \Delta w$$

$$= E(w) - \eta g^T g + \frac{\eta^2}{2} g^T G g < E(w) \quad \textit{for } \eta \textit{ small}$$

Pattern by pattern learning:

$$\Delta w_t = -\eta g_t$$
$$= -\eta g - \eta(g_t - g)$$
$$= \textit{steepest descent} + \textit{stochastic search}$$

Vario Eta Learning: $\Delta w_t = -\frac{\eta}{\sqrt{\frac{1}{T}\sum(g_t - g)^2}} g_t$

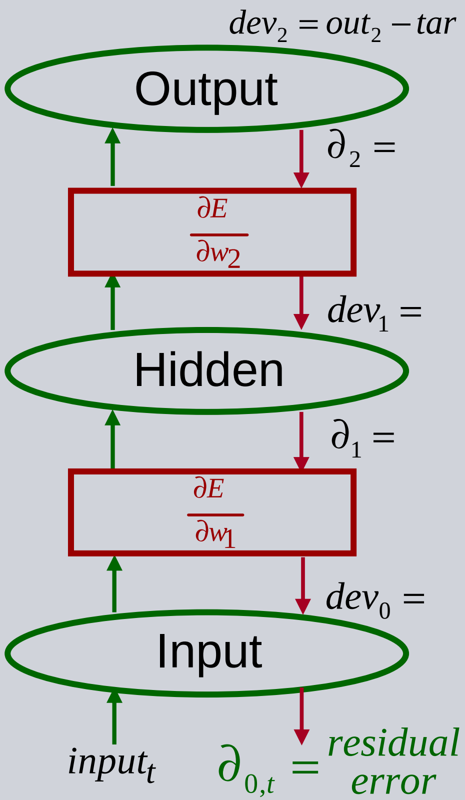



Vario-Eta is a stochastic approx. of the Newton method

SIEMENS

# Data meet Structure:  The Observer - Observation Dilemma

## Calculus of Cleaning




$$x_t = x_t^{data} + clean(\partial_{0,t})$$

$$input_t = x_t^{data} + noise(clean(\partial_{0,t}))$$

**Psychological Dilemma:**
How far should observations determine our picture of the world?
**&**
How far should our picture of the world evaluate observations ?

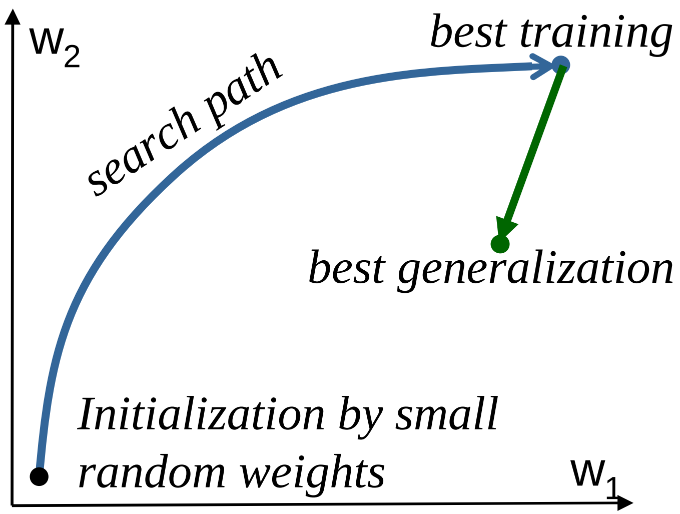
**Technical Dilemma:**
How far should observations determine a model ?
**&**
How far should a model evaluate observations ?

Use the model to clean the data
Data cleaning implies data uncertainty
Use the data uncertainty to harden the learning

# Occams Razor: Search for a Parsimonious Network

**Advise: Late Stopping & Weight Pruning**




Define a criterion for the weight importance:

$$\text{test}_w = w^2$$

**Weight Pruning Procedure:**

Trace the best model

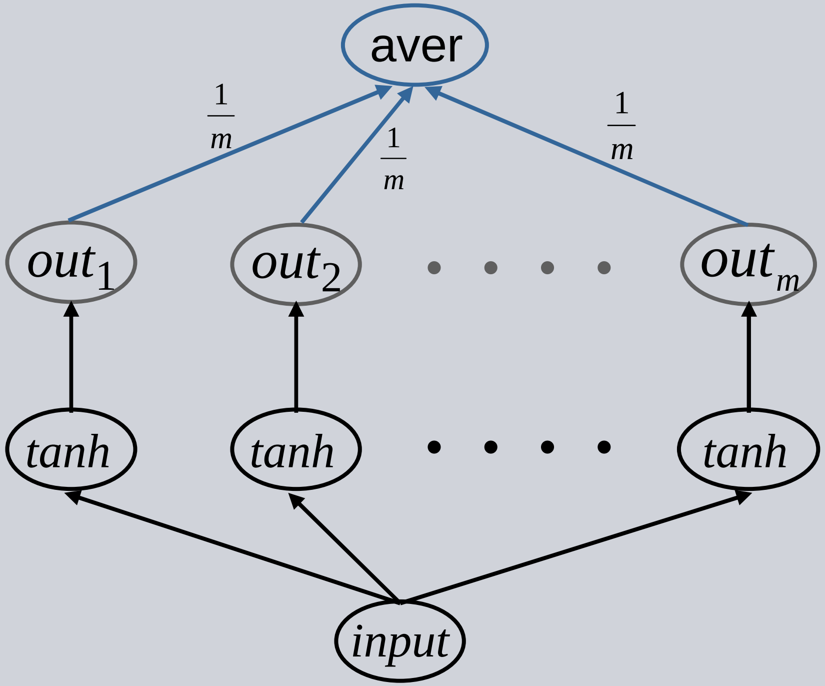
1. Train the Neural Network
2. Rank weights by importance
3. Prune lower ranked weights

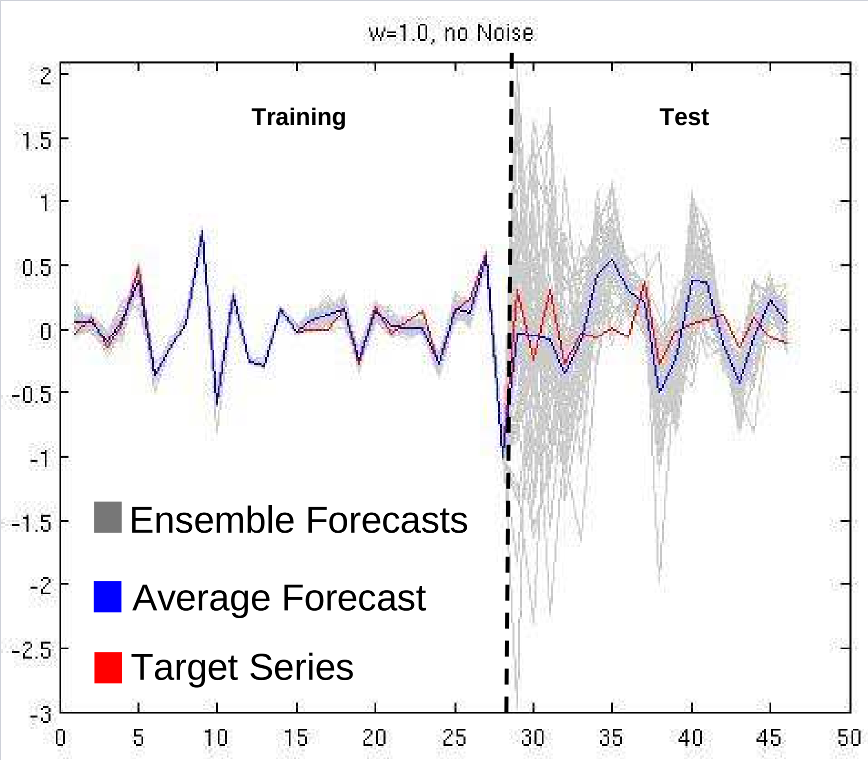
Procedure is bias free towards linear models

Pruning methods split the training data in learning data & validation data, used in the trace.

# Forecasting Expectation & Risk with Ensemble Neural Networks

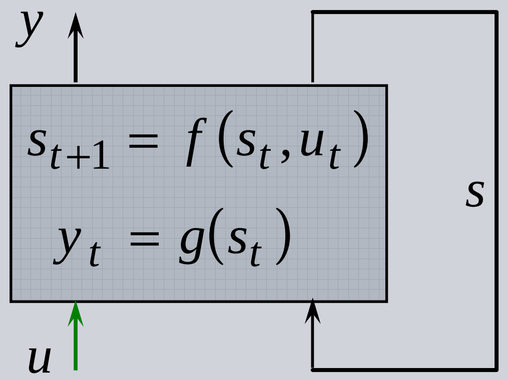
Following Bayes, the expected value of the forecast is computed as the ensemble average.




Uncertainty shows up because we do not know the true scenario. Stochasticity is not seen as a feature of the real world, but as a consequence of partial observability

# Modeling of Open Dynamical Systems with Recurrent Neural Networks (RNN)

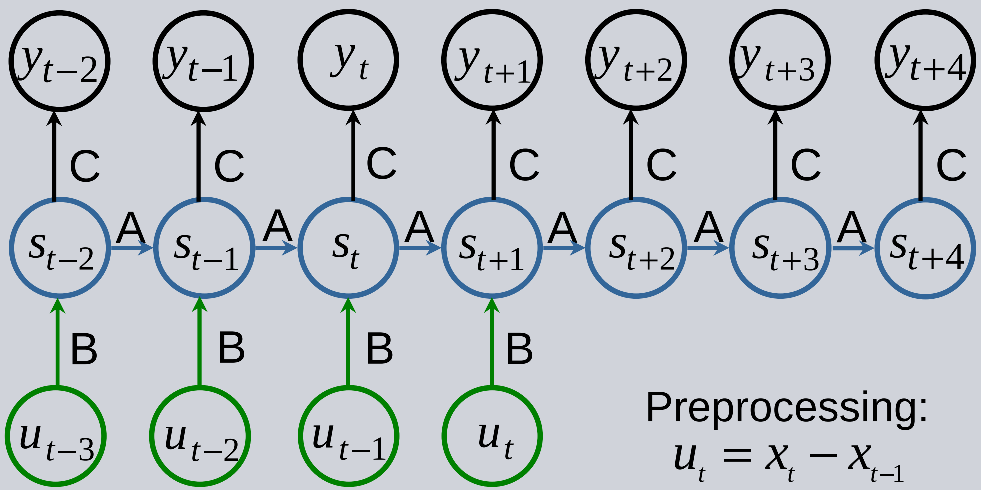



| | |
|---|---|
| $s_{t+1} = \tanh(As_t + Bu_t)$ | state transition |
| $y_t = Cs_t$ | output equation |
| $\sum_{t=1}^{T} (y_t - y_t^d)^2 \to \min\limits_{A,B,C}$ | identification |

*Finite unfolding in time* transforms time into a spatial architecture. We assume, that $x_t$=const in the future.

The analysis of open systems by RNNs allows a decomposition of its autonomous & external driven subsystems.

Long-term predictability depends on the extraction of a strong autonomous subsystem.

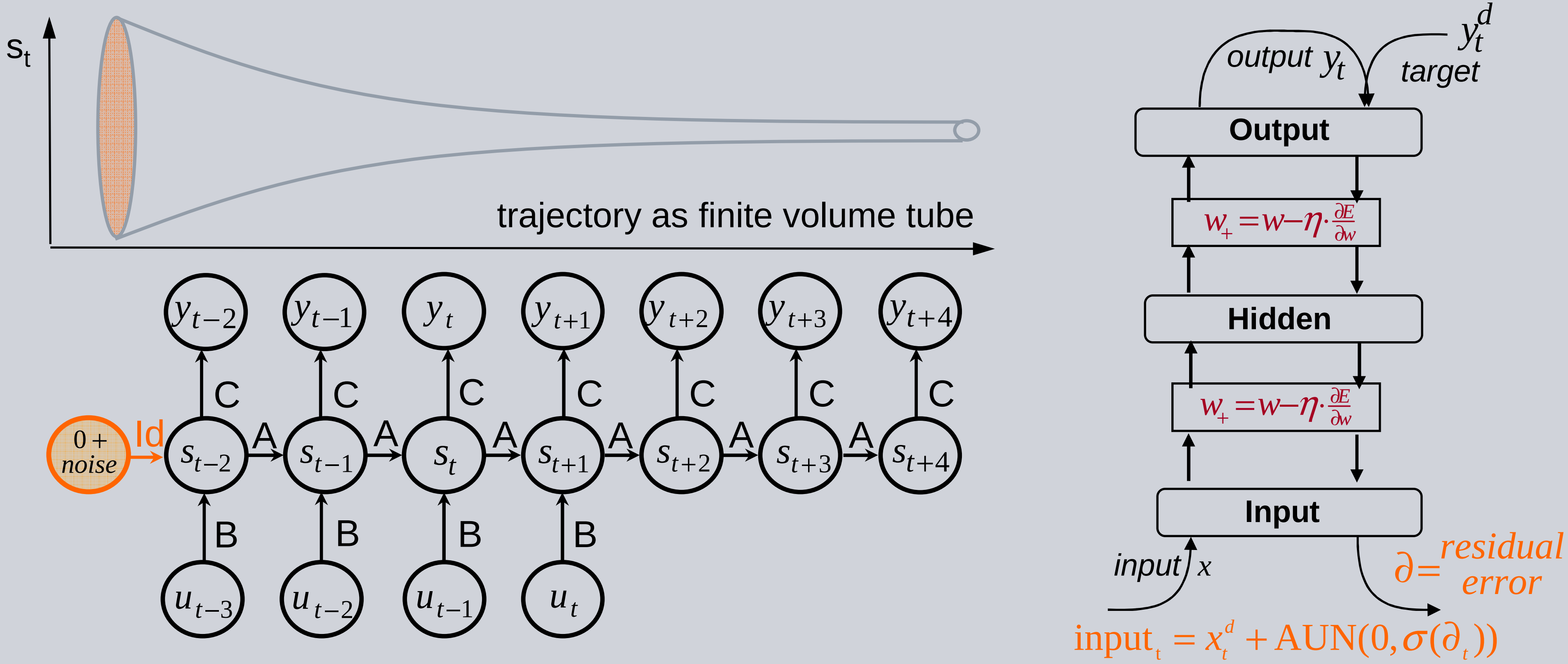



Preprocessing:
$u_t = x_t - x_{t-1}$

SIEMENS

# From Unknown Initial States to Finite Volume Trajectories for RNNs

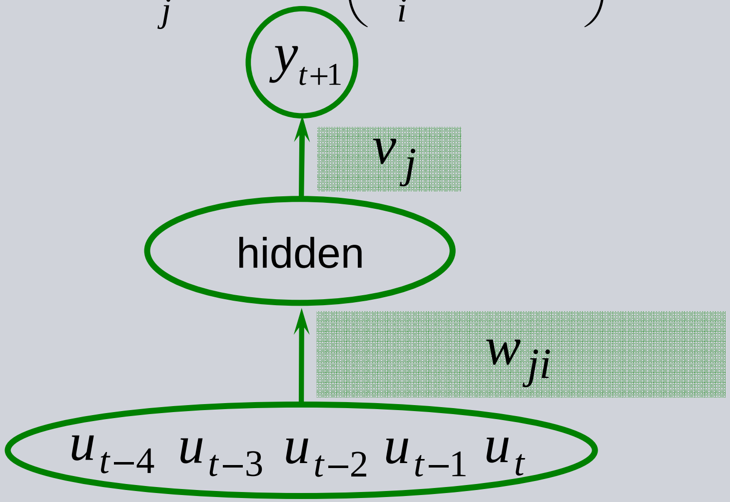
To define a recursion $s_{t+1} = f(s_t, u_t)$ we have to specify an initial value $s_{t-m}$ of the iteration!



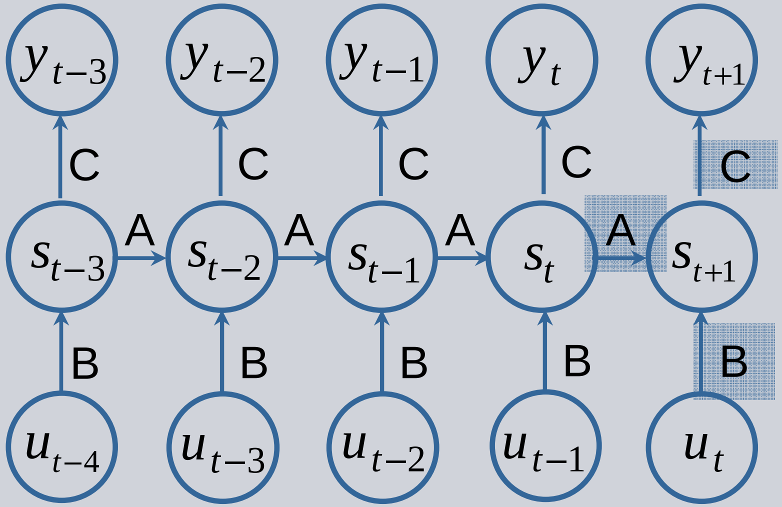

- By initial noise (**A**daptive **U**niform **N**oise) we harden the model against the unknown $s_{t-m}$ .
- Matrix A becomes a contraction, squeezing out the initial uncertainty.

## Feedforward versus Recurrent Neural Networks Architectures

$$y_{t+1} = \sum_j v_j \tanh\left(\sum_i w_{ji} u_{t-i}\right)$$



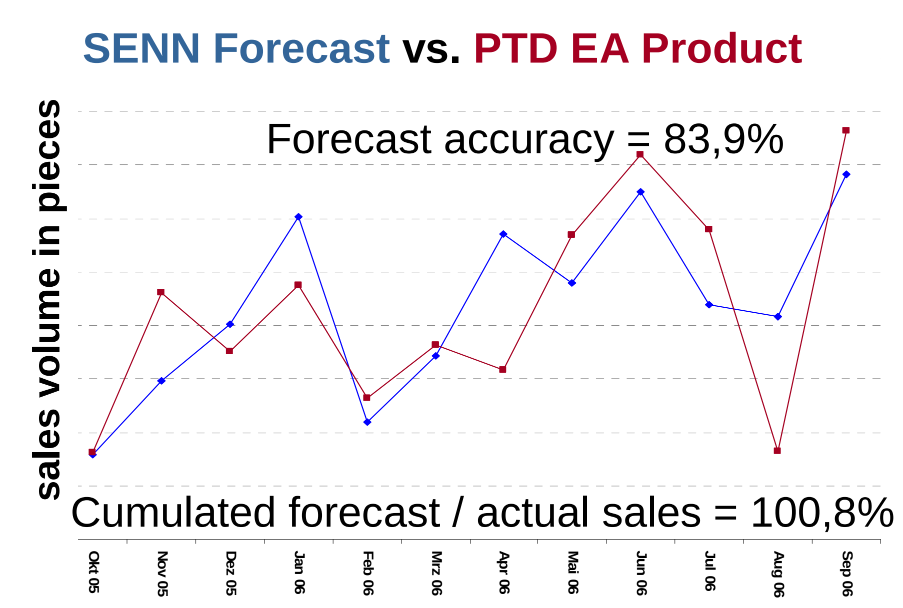

$$s_{t+1} = \tanh(As_t + Bu_t) \quad , \quad y_t = Cs_t$$




- Unfolding in time allows the representation of a temporal process in form of a recurrent neural network, if the matrices A, B and C are constant over time.
- In contrast to a feedforward network, the recurrent structure depends on fewer parameters and provides more gradient information.
- Each vertical branch of the recurrent net is a 3-layer MLP.

SIEMENS

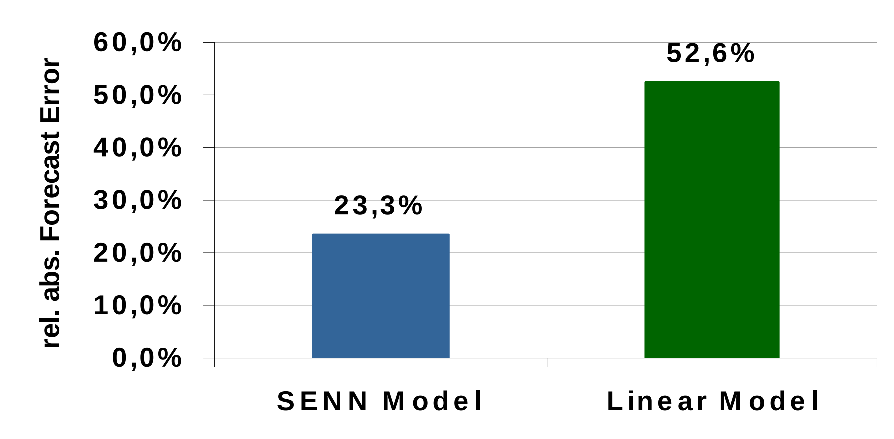
# Recurrent Neural Networks in Demand Forecasting




| Expert Group | Forecast Error SENN Model | Linear Model |
|---|---|---|
| 1 | 8,6% | 10,4% |
| 2 | 42,6% | 71,9% |
| 3 | 13,8% | 36,7% |
| 4 | 10,4% | 24,1% |
| 5 | 27,7% | 27,3% |
| 6 | 15,0% | 21,7% |
| 7 | 24,0% | 27,4% |
| 8 | 16,9% | 121,5% |
| 9 | 26,9% | 39,5% |
| 10 | 26,1% | 22,3% |
| 11 | 23,8% | 35,8% |
| 12 | 35,7% | 60,4% |
| 13 | 51,3% | 237,9% |
| 14 | 12,2% | 12,3% |
| 15 | 19,5% | 29,2% |
| 16 | 18,1% | 63,0% |
| Average | 23,3% | 52,6% |
| Std. Deviation | 11,5% | 54,9% |

Neural networks clearly outperform linear regression models:

The forecast accuracy of the neural networks is higher and more stable.



SIEMENS

## From Recurrent Neural Networks to Error Correction Neural Networks

An error correction system considers the forecast error in present time as a reaction on unknown external information.

$$
\begin{aligned}
s_{t+1} &= f(s_t, u_t) \\
y_t &= g(s_t)
\end{aligned}
$$

$y$, $u$, $s$

$\longrightarrow$

$$
\begin{aligned}
s_{t+1} &= f\left(s_t, u_t, (y_t - y_t^d)\right) \\
y_t &= g(s_t)
\end{aligned}
$$

$y$, $u$, $s$

In order to correct the forecasting this error is used as an additional input, which substitutes the unknown external information.

# Handling Shocks with Error Correction Neural Networks

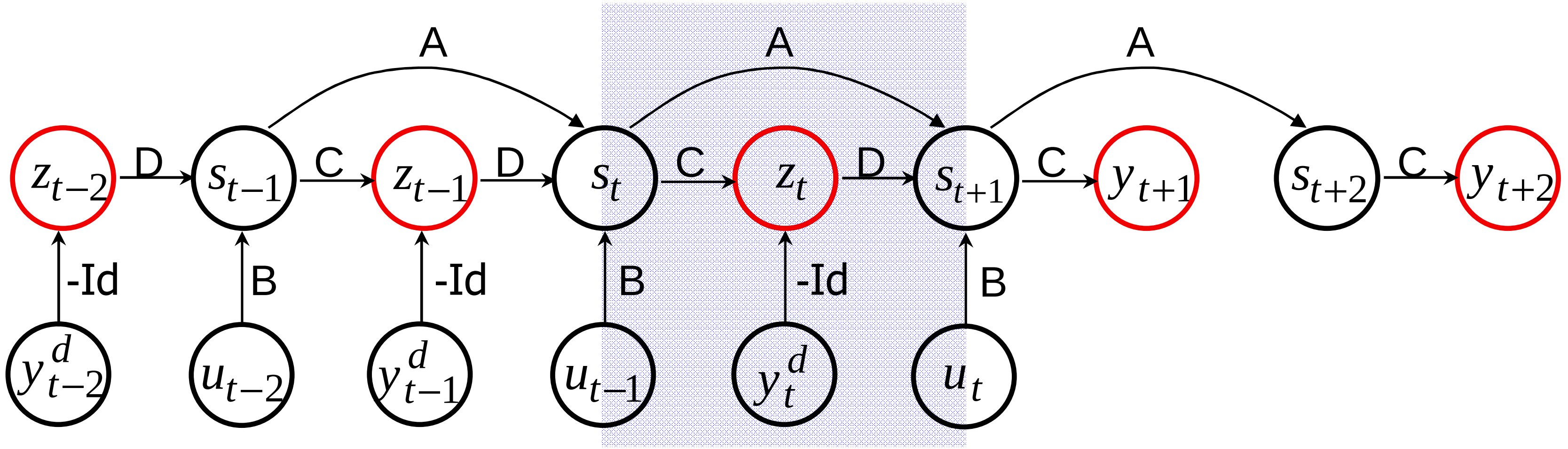
SIEMENS

An error correction system considers the forecast error as a measurement of
ex ante unknown additional external information or external system shocks.

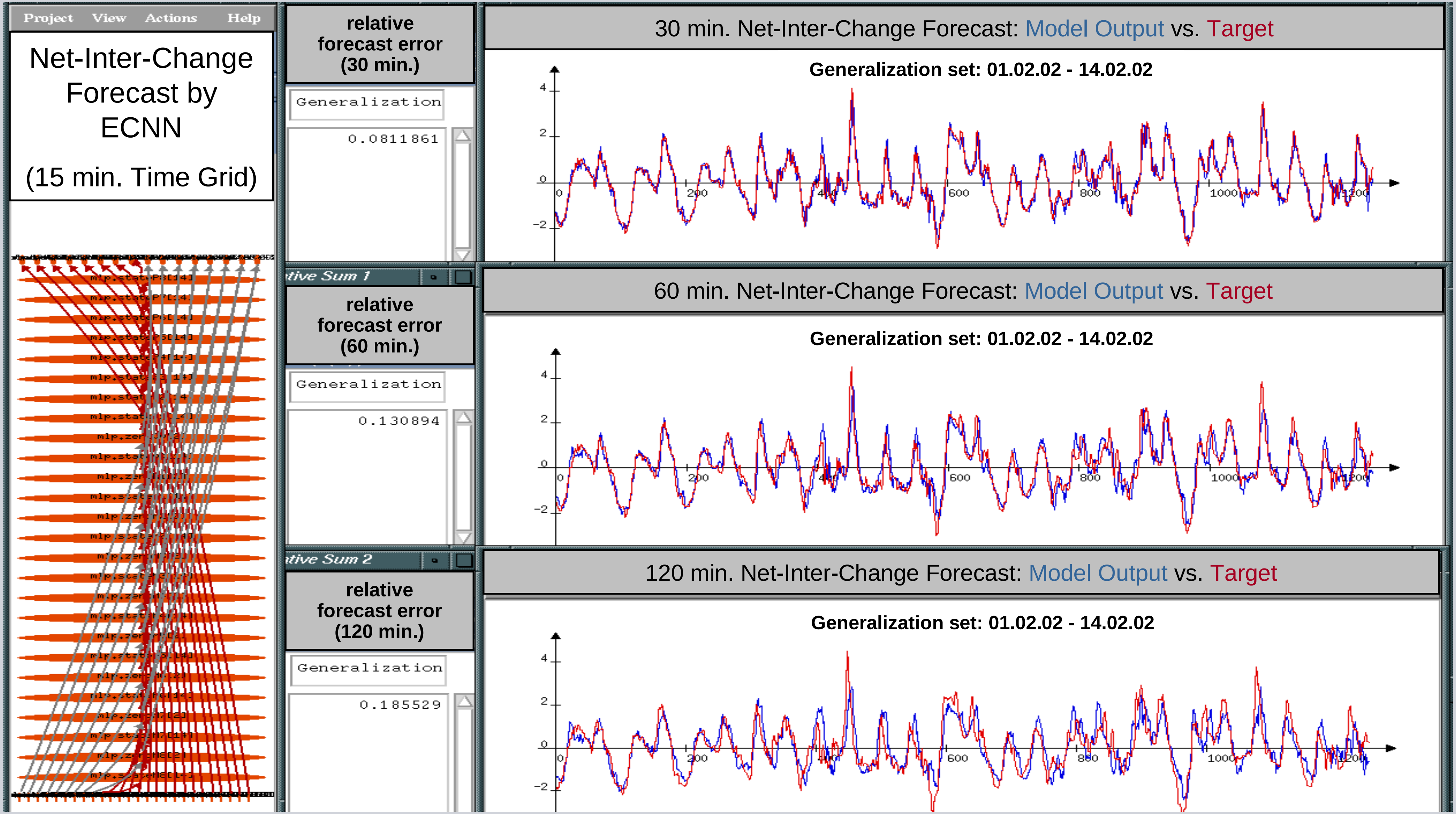
$$s_{t+1} = f\left(s_t, u_t, (y_t - y_t^d)\right)$$ **state transition**

$$y_t = g(s_t)$$ **output equation**

$$\frac{1}{T}\sum_{t=1}^{T}\left(y_t - y_t^d\right)^2 \to \min_{f,g}$$ **identification**

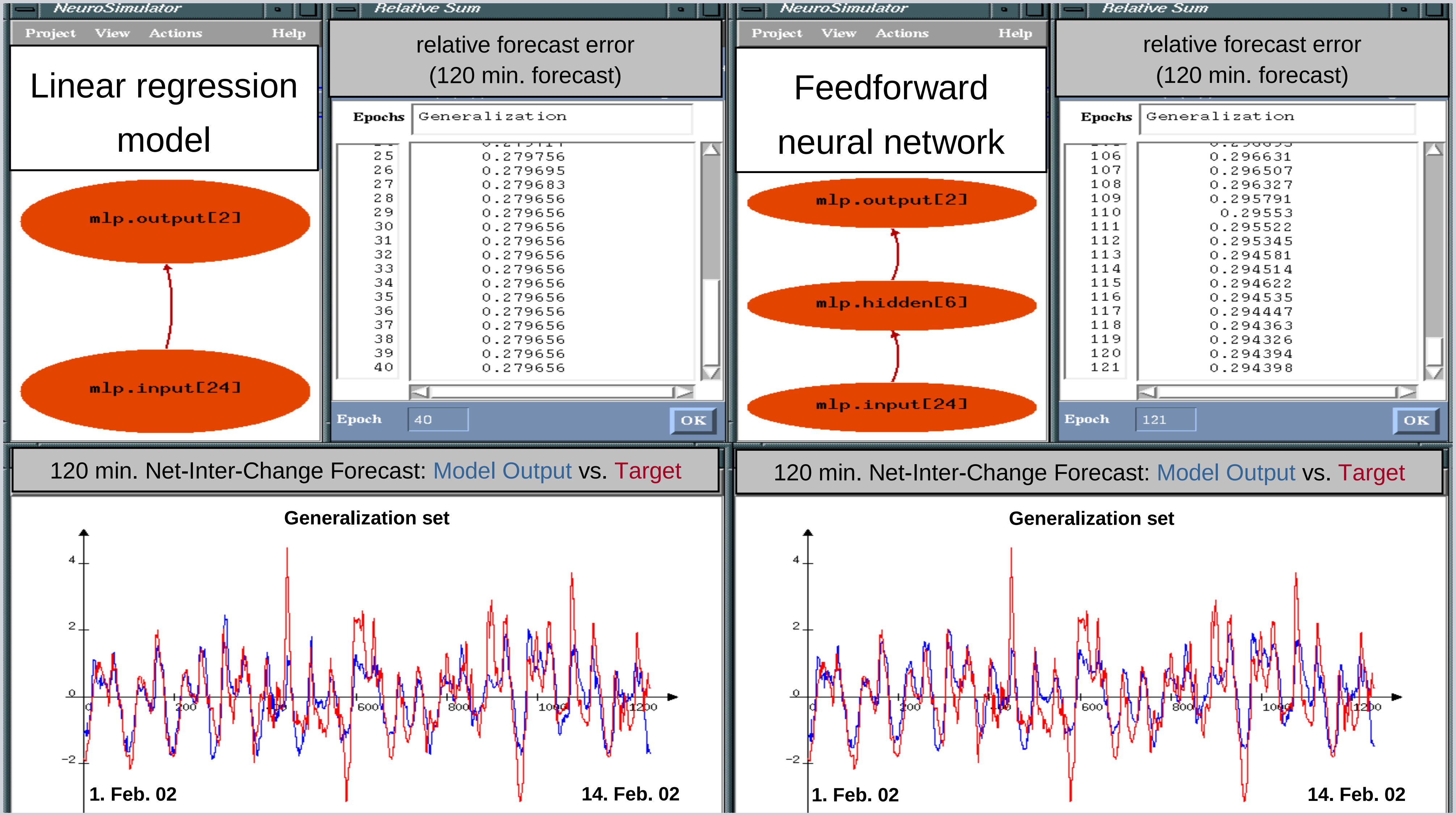


# Net Interchange Forecast with Error Correction Neural Networks



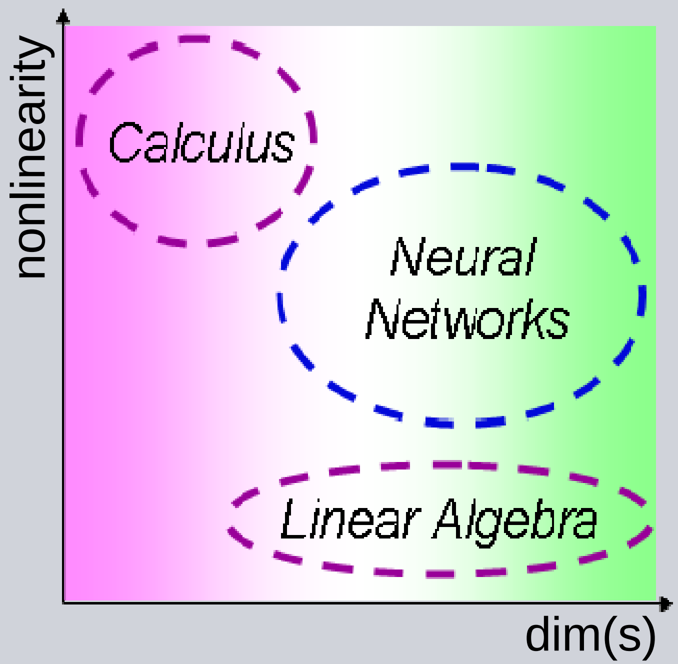
# Net Interchange Forecast with Linear Regression & Feedforward Networks



SIEMENS

## Large Recurrent Neural Networks are Special



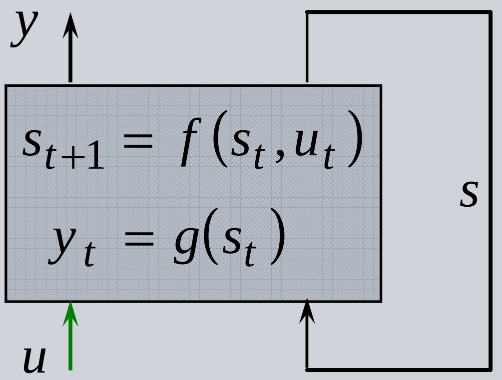

- In case of a changing environment, large networks have to be dynamical consistent closed systems.
- For changing environments large open systems would define an inconsistent learning task.

- To avoid signal avalanches, large recurrent networks have to be sparsely interconnected.
- Random sparsity improves long term memory and supports the modeling of multiple time scales.

- Large recurrent networks model the data perfectly, still leaving the opportunity of an eigendynamics.
- The eigenactivity can be seen as a self-created internal noise, which acts as a self-regularization.

# Dynamical Consistency Problem in the Modeling of Dynamical Systems



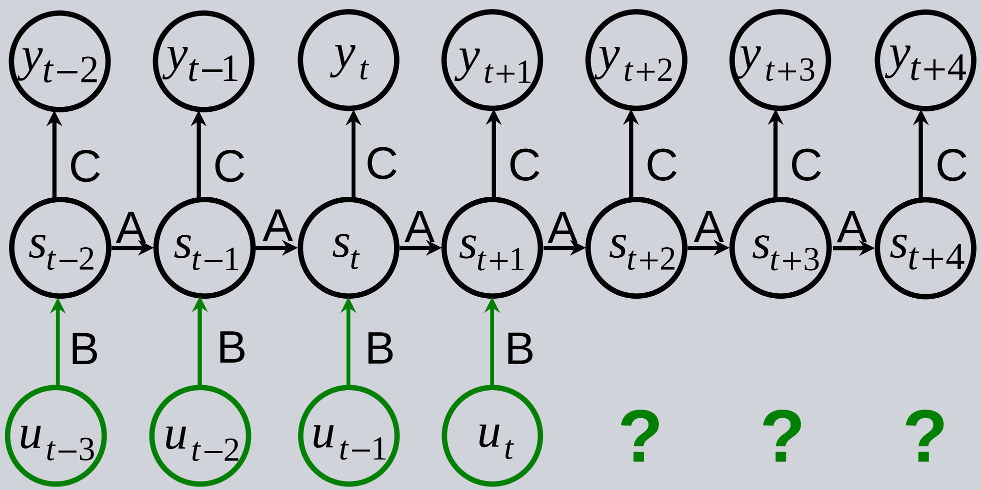

$$s_{t+1} = \tanh(As_t + Bu_t)$$ state transition

$$y_t = Cs_t$$ output equation

$$\sum_{t=1}^{T} (y_t - y_t^d)^2 \to \min_{A,B,C}$$ identification

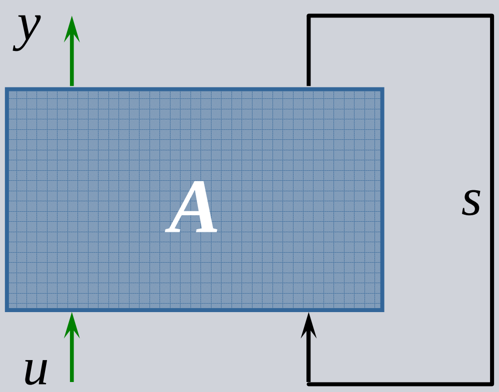
*Modeling in form of open dynamical systems is only consistent if the external drivers are nearly constant from present time on.*

Otherwise the learning works only for small RNNs because, following the regression paradigm, the optimization finds an intermediate solution between the active & the constant environment.



## From Standard to Normalized Recurrent Neural Networks

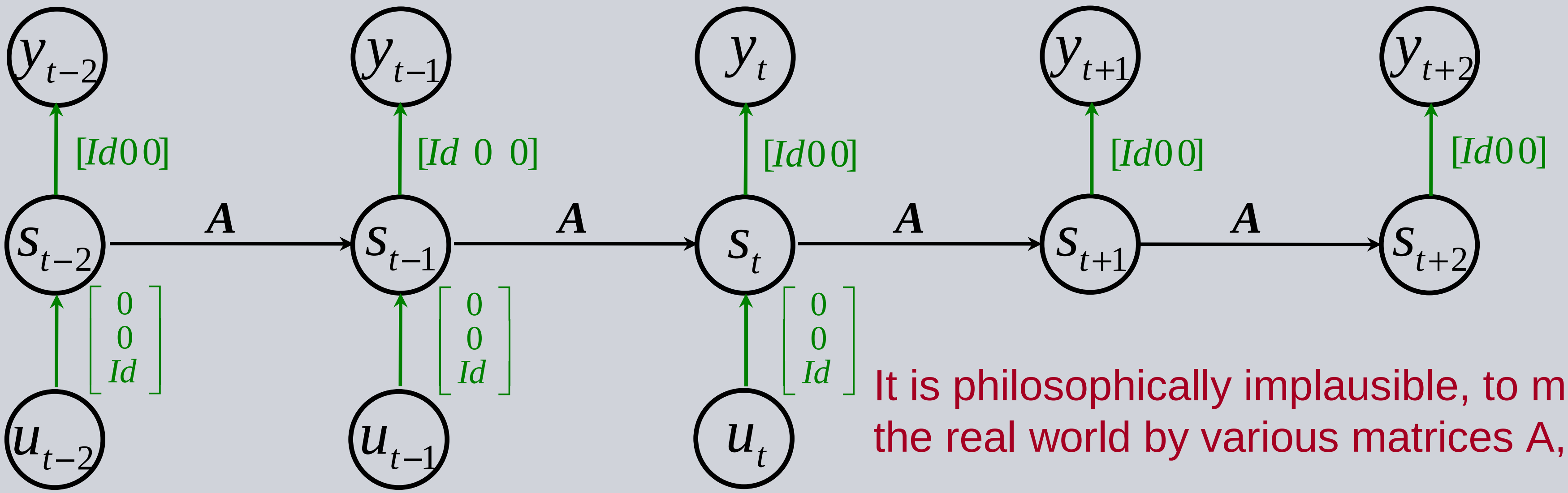



W.l.o.g. we can model a dynamics by one matrix A:

$$\tau \le t: \; s_\tau = \tanh\left(A s_{\tau-1} + \begin{bmatrix} 0 \\ 0 \\ \mathrm{Id} \end{bmatrix} u_\tau \right)$$

$$\tau > t: \; s_\tau = \tanh(A s_{\tau-1})$$

$$y_\tau = [\mathrm{Id} \;\; 0 \;\; 0] \, s_\tau \quad , \quad \sum_{t=m}^{T-n} \sum_{\tau=t-m}^{t+n} (y_\tau - y_\tau^d)^2 \to \min_A$$




It is philosophically implausible, to model the real world by various matrices A, B, C.

## Modeling the Dynamics of Observables

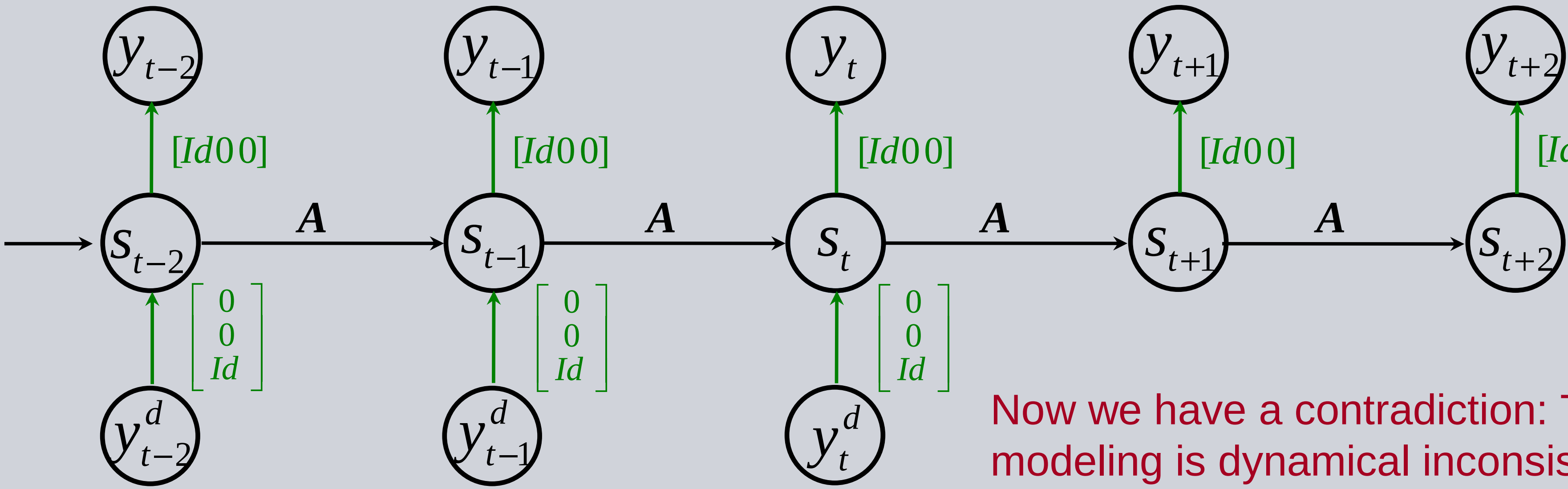
Inputs and targets are merged to observables. In the net we refer to observations $y_t^d$ and expectations $y_t$ of these observables.

Due to the network design, the input to output relation is delayed.

$$\tau \le t: \quad s_\tau = \tanh\left(A s_{\tau-1} + \begin{bmatrix} 0 \\ 0 \\ \mathrm{Id} \end{bmatrix} y_\tau^d \right)$$

$$\tau > t: \quad s_\tau = \tanh(A s_{\tau-1})$$

$$y_\tau = \begin{bmatrix} \mathrm{Id} & 0 & 0 \end{bmatrix} s_\tau \quad , \quad \sum_{t=m}^{T-n} \sum_{\tau=t-m}^{t+n} (y_\tau - y_\tau^d)^2 \to \min_A$$




Now we have a contradiction: The modeling is dynamical inconsistent.

# Modeling Dynamical Systems with Dynamical Consistent Neural Networks

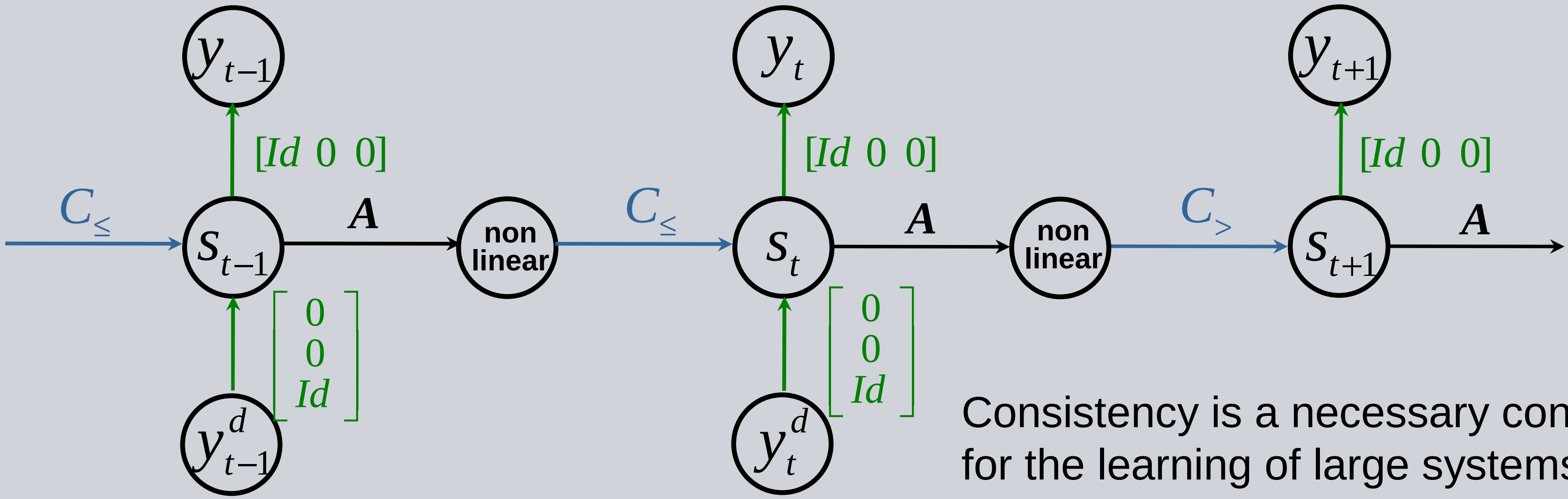
Missing future observations are substituted by the models own expectations.

DCNNs overcome the schizophrenic learning of only 1 matrix A for both a changing and a constant environment.

$$\tau \le t: \quad s_\tau = \underbrace{\begin{bmatrix} Id & 0 & 0 \\ 0 & Id & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{=C_\le} \tanh\left(A s_{\tau-1}\right) + \begin{bmatrix} 0 \\ 0 \\ Id \end{bmatrix} y_\tau^d = \begin{pmatrix} y_\tau\text{: expectations} \\ h_\tau\text{: hidden var} \\ y_\tau^d\text{: observations} \end{pmatrix}$$

$$\tau > t: \quad s_\tau = \underbrace{\begin{bmatrix} Id & 0 & 0 \\ 0 & Id & 0 \\ Id & 0 & 0 \end{bmatrix}}_{=C_>} \tanh\left(A s_{\tau-1}\right) = \begin{pmatrix} y_\tau\text{: expectations} \\ h_\tau\text{: hidden var} \\ y_\tau\text{: expectations} \end{pmatrix}$$

$$all\ \tau: \quad y_\tau = \begin{bmatrix} Id & 0 & 0 \end{bmatrix} s_\tau \qquad \sum_{t=m}^{T-n} \sum_{\tau=t-m}^{t+n} (y_\tau - y_\tau^d)^2 \to \min_A$$

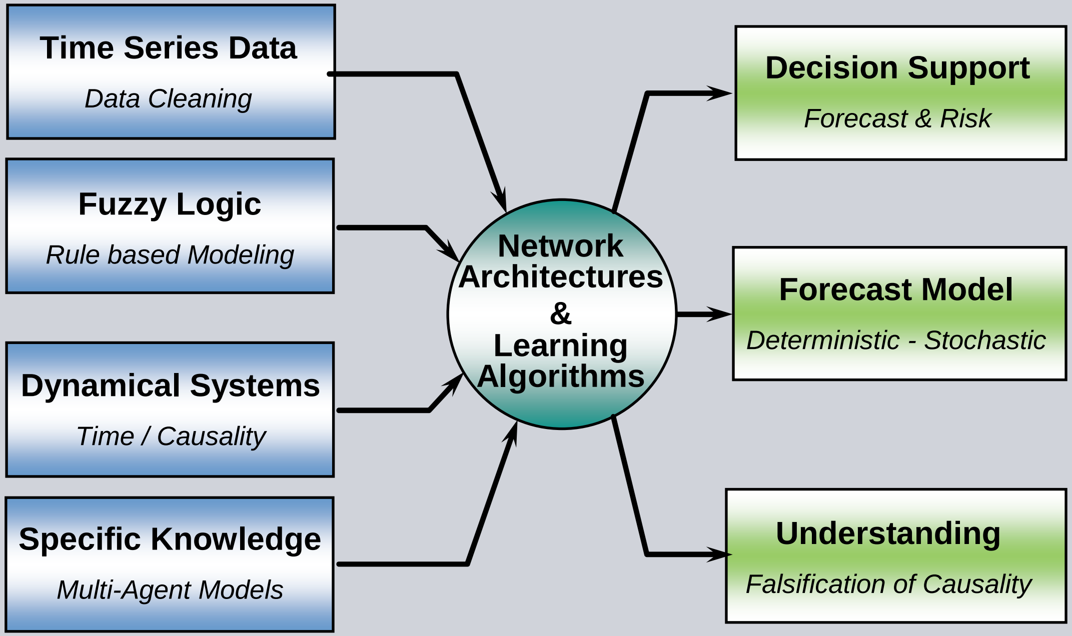



Consistency is a necessary condition for the learning of large systems.

SIEMENS

# Neural Networks - From Data Mining to Model Building

Data alone often do not cover the modeling task. Thus, we merge model building by data, prior knowledge and first principles.




Neural networks (**SENN**) allow systems analysis, forecasting & risk analysis as well as the setup of decision support systems.