



Recurrent neural networks RNN/LSTM

Sequence Prediction

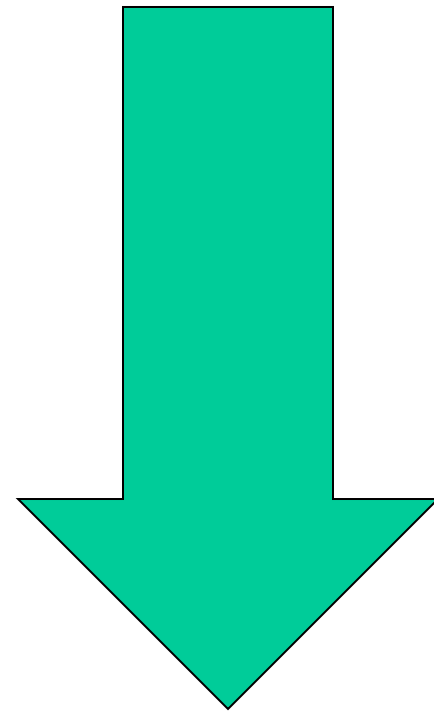
Neurocomputing
Teil 3



Sequenzen

Sequenz Daten:

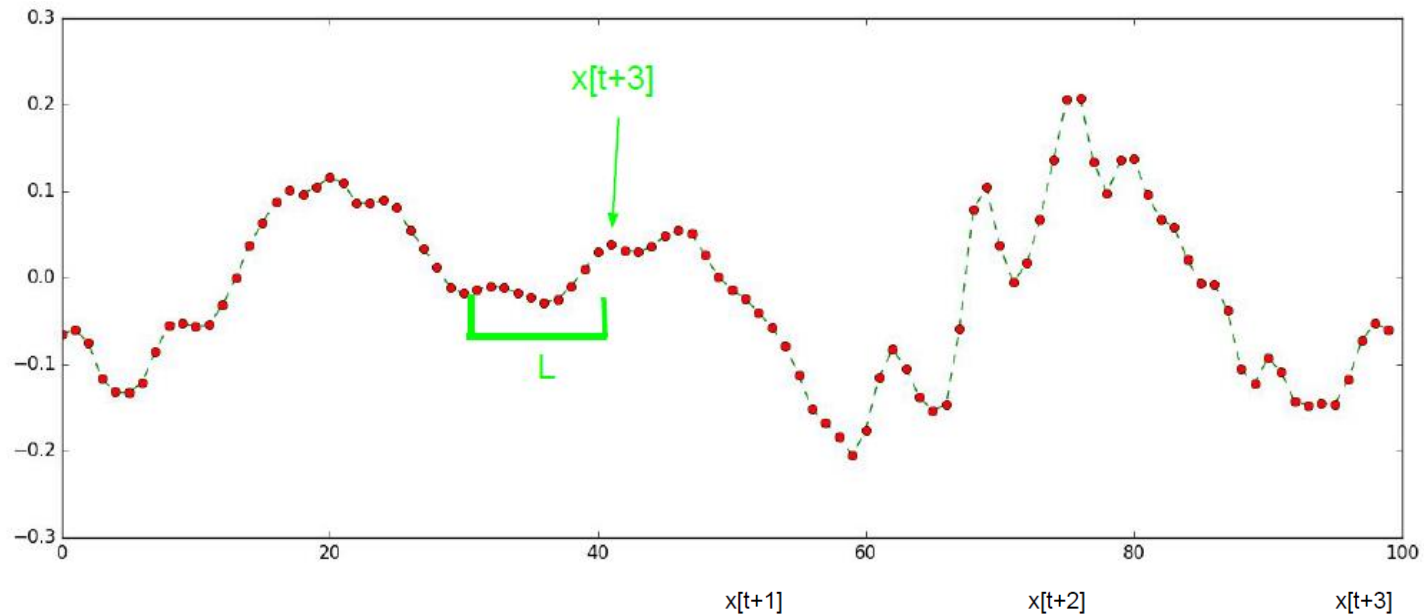
- Zeitreihen (NARX-Netz)
- Texte: Wort Sequenzen
- NLP Sequenzen
- Audiosequenzen



RNN

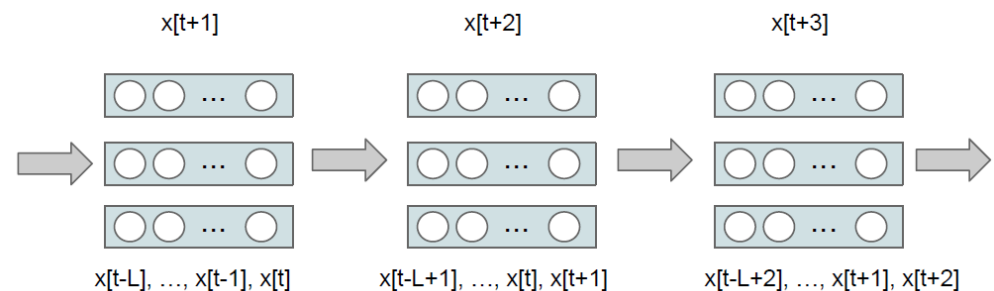


Sequence Prediction und Feed Forward MLP

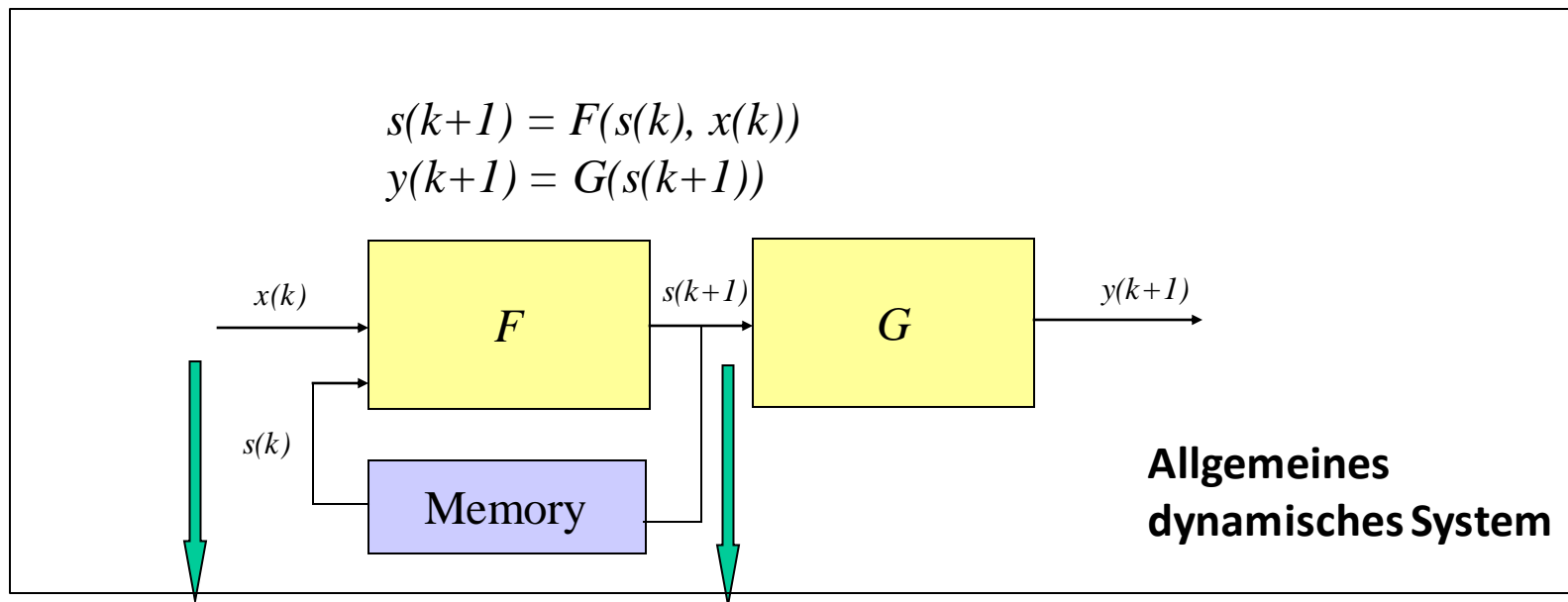


Feed Forward approach:

- static window of size L
- slide the window time-step wise across input data x



Dynamisches System: Rückgekoppeltes NARX-Netz



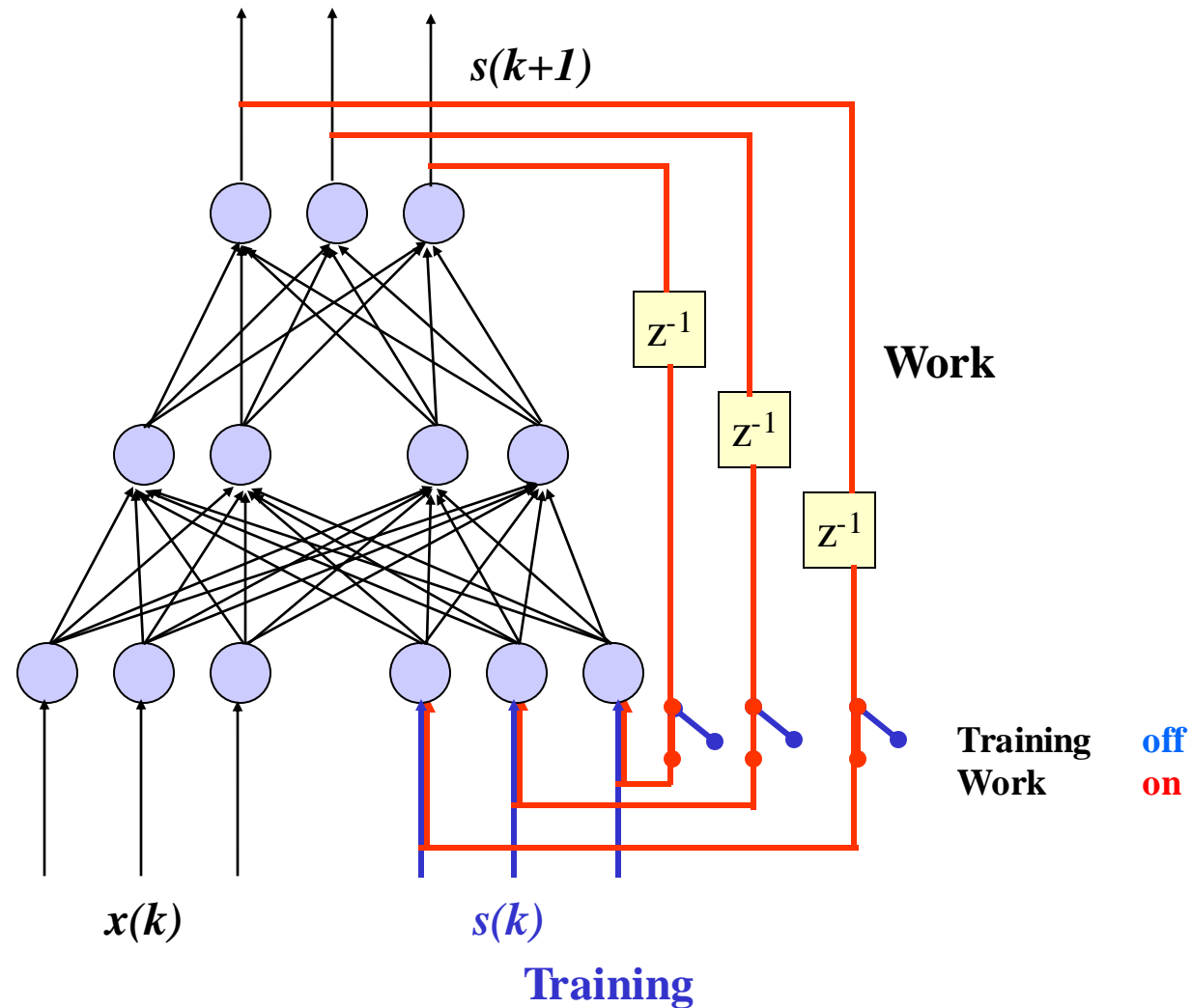
Experiment

$$E = \{(\mathbf{s(1)}, x(0)), (\mathbf{s(2), x(1)}), (s(3), x(2)), (s(4), x(3)), \dots, (s(n), x(n-1))\}$$

$$P = \{(\mathbf{s(2), (s(1), x(1))}), (s(3), (s(2), x(2))), (s(4), (s(3), x(3))), \dots, (s(n), (s(n-1), x(n-1)))\}$$

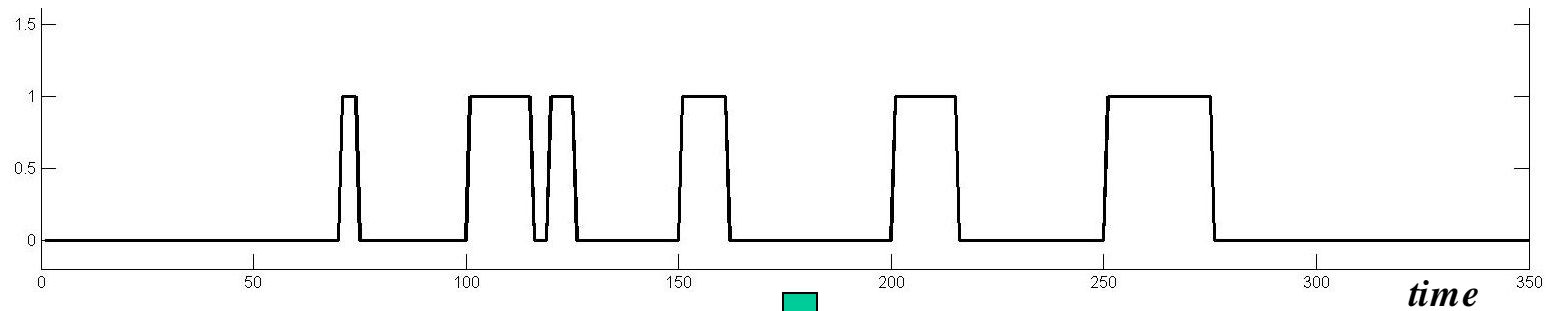
Pattern zum Trainieren

Dynamisches System - Dynamisches Neuronales Netz (Rückgekoppeltes NARX-Netz)





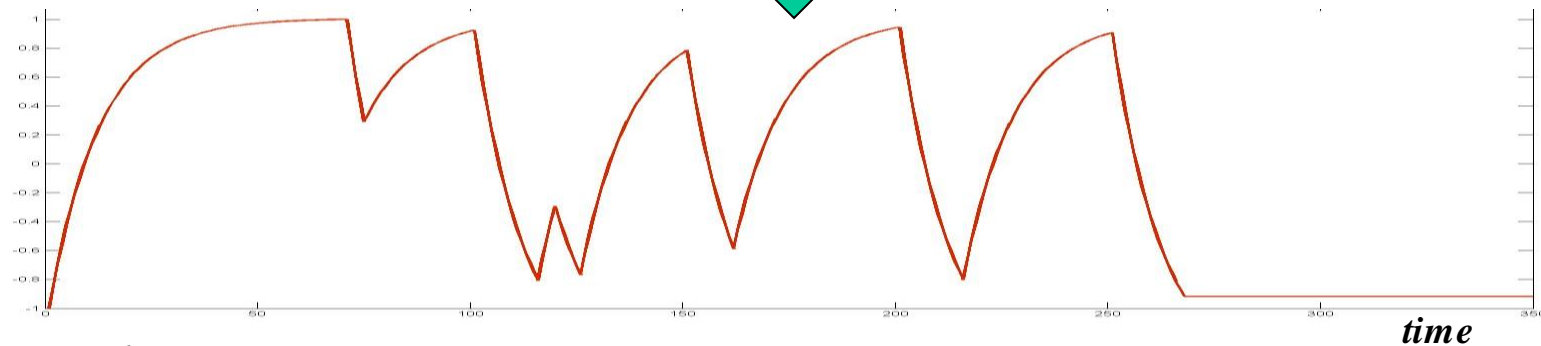
Beispiel: Dynamisches System



$x(k)$ input

$$s(k+1) = F(s(k), x(k))$$

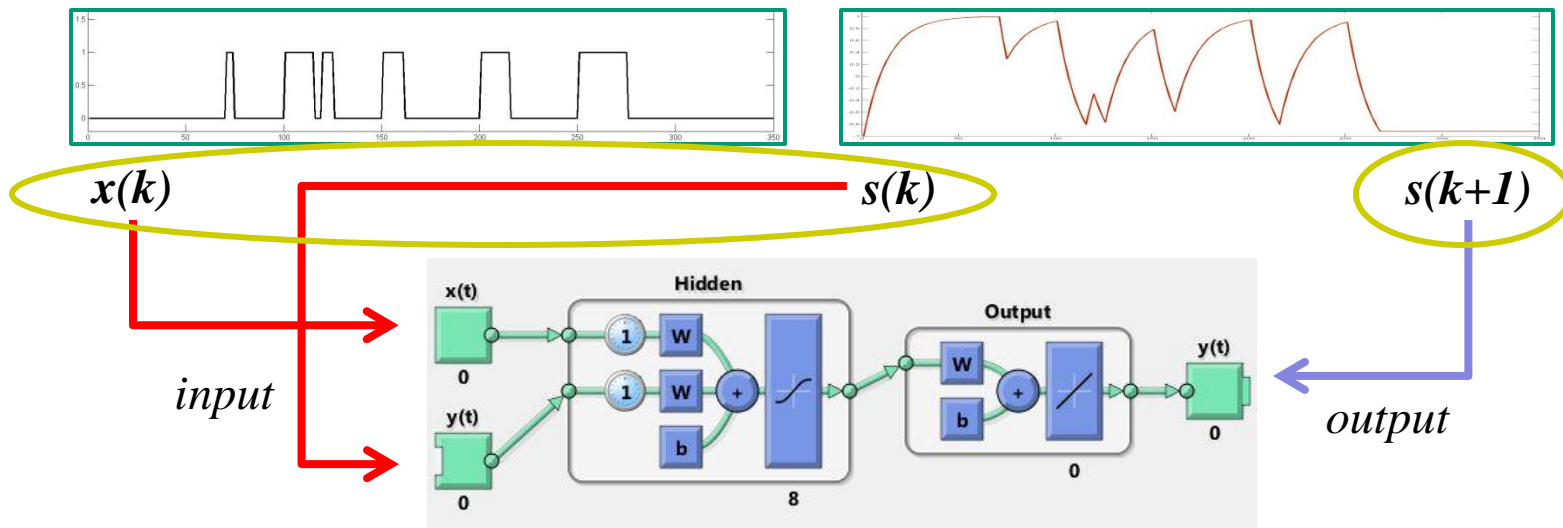
Dynamical system



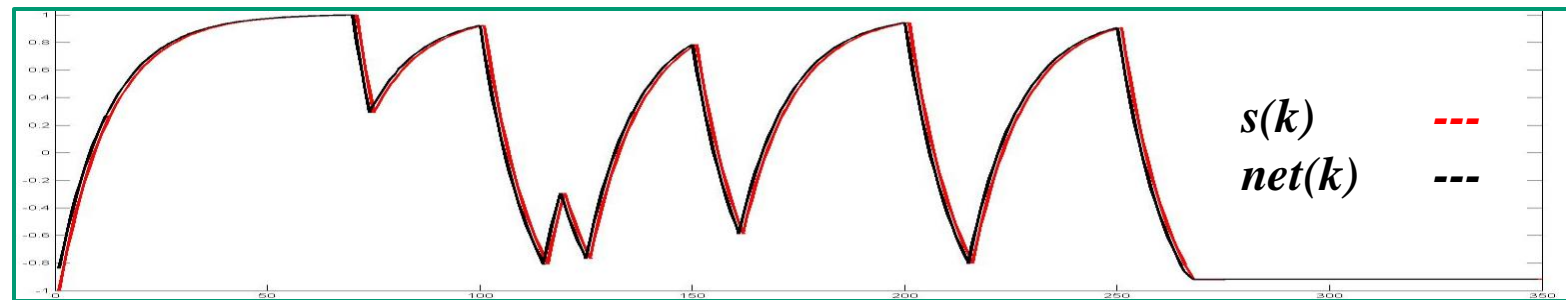
$s(k)$ state - output



Dynamisches Neuronales Netz: Training



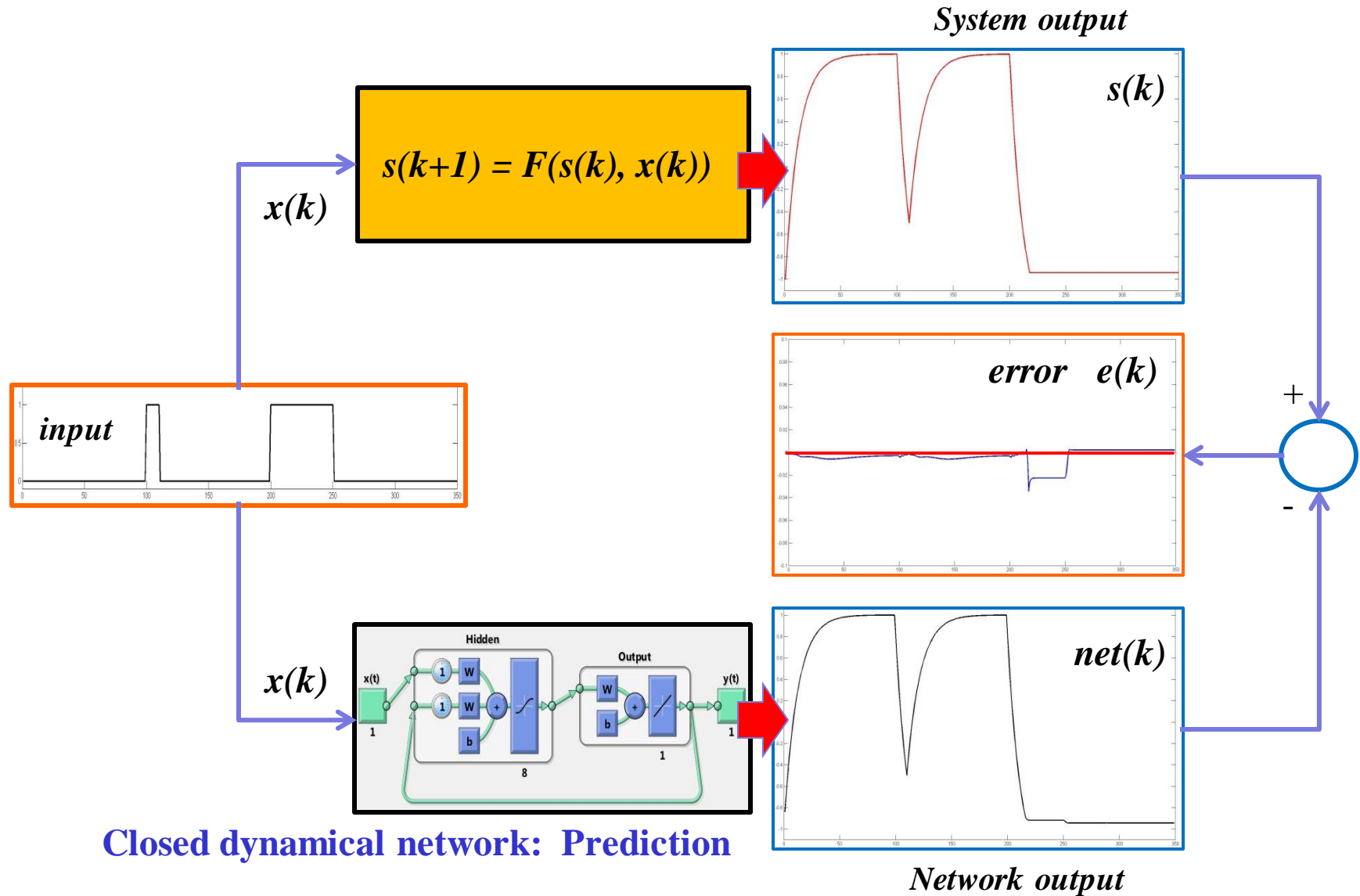
Open dynamical network: Training



Comparison: Network output (black) vs. system output (red)



Dynamisches Neuronales Netz: Prediction



Closed dynamical network: Prediction



RNNs

Recurrent neural networks (RNN):

Rekurrente Neuronale Netze (RNN) sind Feedforward Neuronale Netze, die durch die Einbeziehung von Übergängen, die aneinander angrenzende Zeitschritte überspannen und einen Zeitablauf einführen.

Wie Feedforward-Netzwerke haben RNNs keine Zyklen, jedoch Übergänge, die angrenzende Zeitschritte miteinander verbinden und eine Ablauffolge definieren.

x : input

y : output


h : internal state (memory of the network)

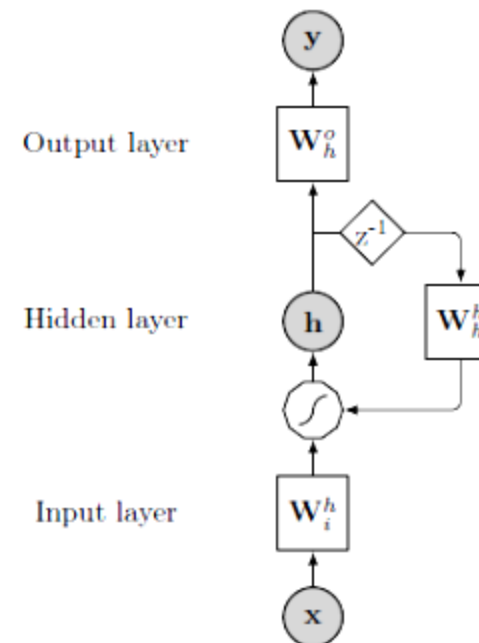
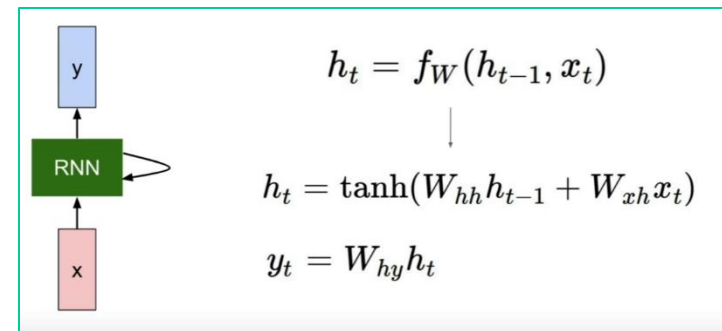
W_i^h : input weights

W_h^h : recurrent layer weights

W_h^o : output weights

z^{-1} : time-delay unit

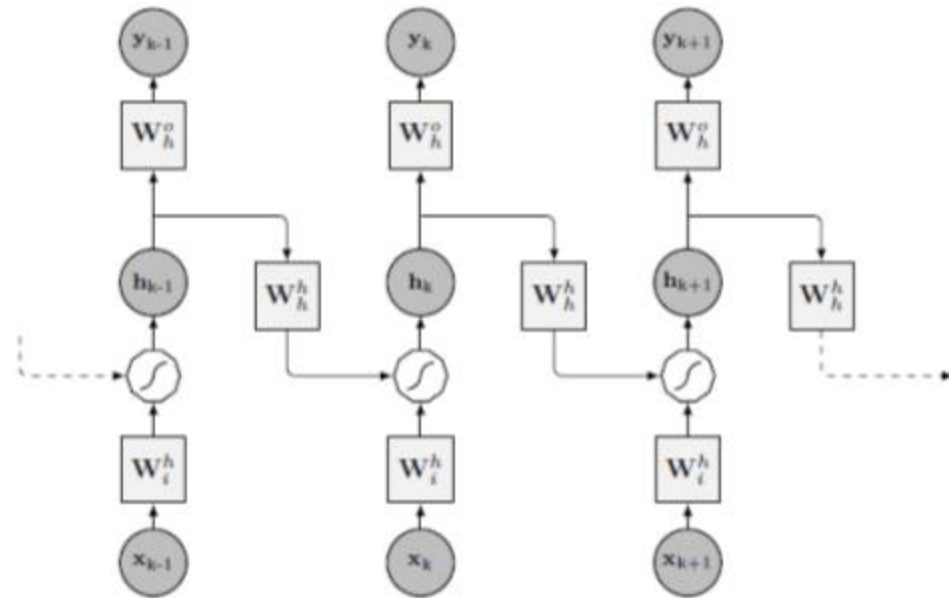
 : neuron transfer function



Einfaches RNN

Backpropagation in RNN: Aufrollen des RNNs

- Um das RNN zu trainieren, muss es aufgerollt werden, dadurch wird es in eine Art Feed Forward-NN umgewandelt.
- Das Netzwerk wird für jeden Zeitschritt kopiert.
- Dann kann Backpropagation angewendet werden. Für RNNs wird Backpropagation Through Time (BPPT) eingesetzt.
- Durch das Kopieren wird das aufgerollte Netzwerk sehr tief!
- Die Gewichte \mathbf{W}_i^h , \mathbf{W}_h^h , \mathbf{W}_h^o sind nach jedem Trainingsschritt im gesamten aufgerollten Netzwerk anzupassen.





Problem Vanishing Gradient (verschwindender Gradient): *Zuviele Multiplikationen*

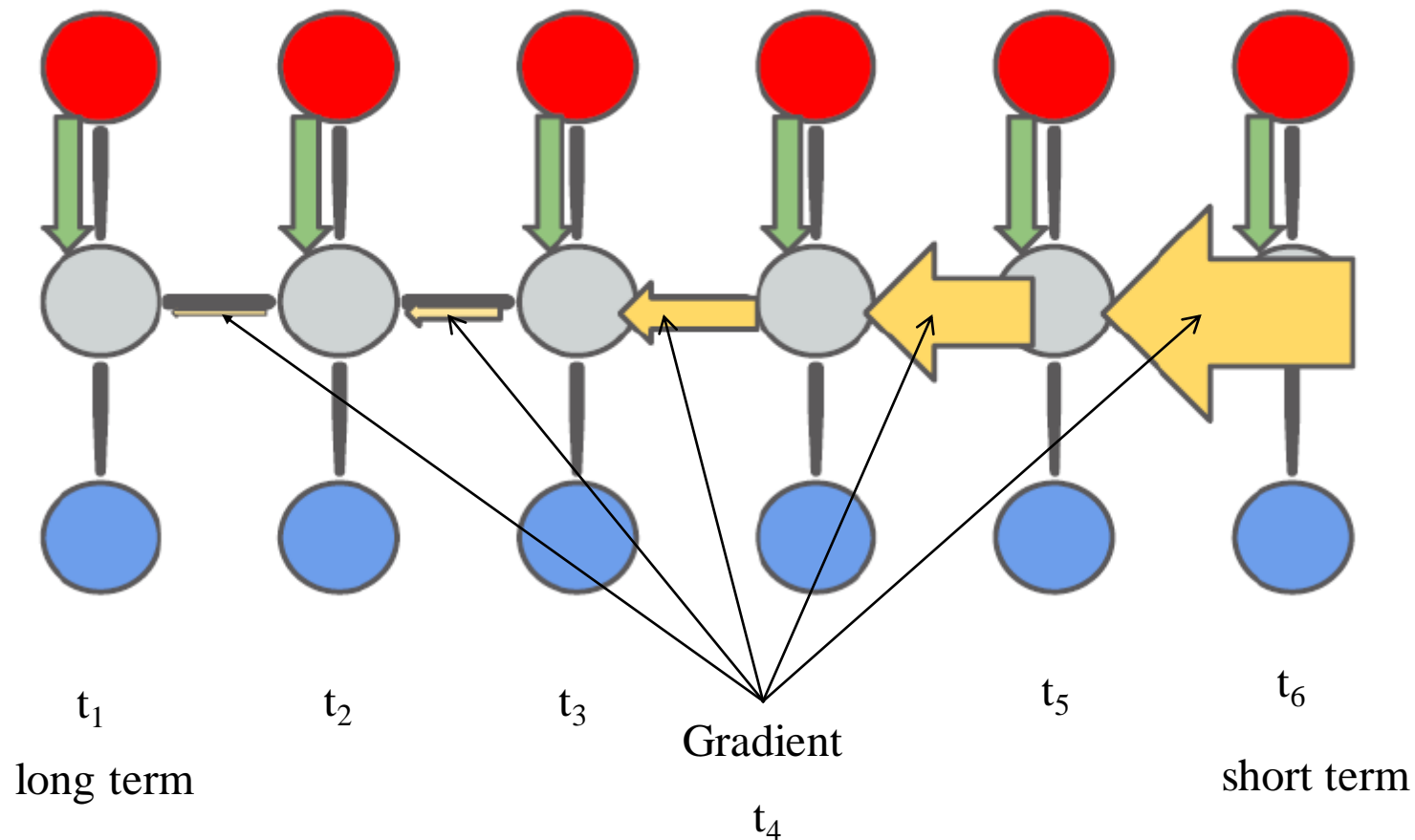
- Jedes Gewicht des aufgerollten RNNs bekommt ein Update proportional zum Gradienten der Errorfunktion am letzten Outputneuron des Zeitschrittes. Diese Updates erfolgen multiplikativ über die Kettenregel (backpropagation)
- Je tiefer das RNN aufgerollt wird, desto mehr Ableitungen der Aktivierungsfunktion müssen miteinander multipliziert werden
- Sind diese während des Trainings <1 , dann multiplizieren sich mehrere Faktoren zu einer Zahl, die schnell gegen Null strebt, weswegen die Gewichtsänderungen in tiefen Schichten deutlich langsamer sind als die in höheren.
- Da zumeist der ***tanh(x)*** als Aktivierungsfunktion verwendet wird, führt dies unausweichlich zu diesem Problem, denn alle Funktionswerte und auch die Ableitungen an allen Stellen sind stets betragsmäßig <1 .

$$w_{ij}^{neu} = w_{ij}^{alt} - \alpha \underbrace{\partial E / \partial w_{ij}}_{\sim 0}$$

Das Netzwerk hört auf zu lernen!

Vanishing Gradient

- Starker **short term gradient**: gutes Kurzzeitgedächtnis
- Verschwindender **long term gradient**: schlechtes Langzeitgedächtnis
RNN „vergisst“ längere vergangene Schritte schnell
RNN kann maximal ca. 10 Zeitschritte erfolgreich lernen





Einführung von Speicher

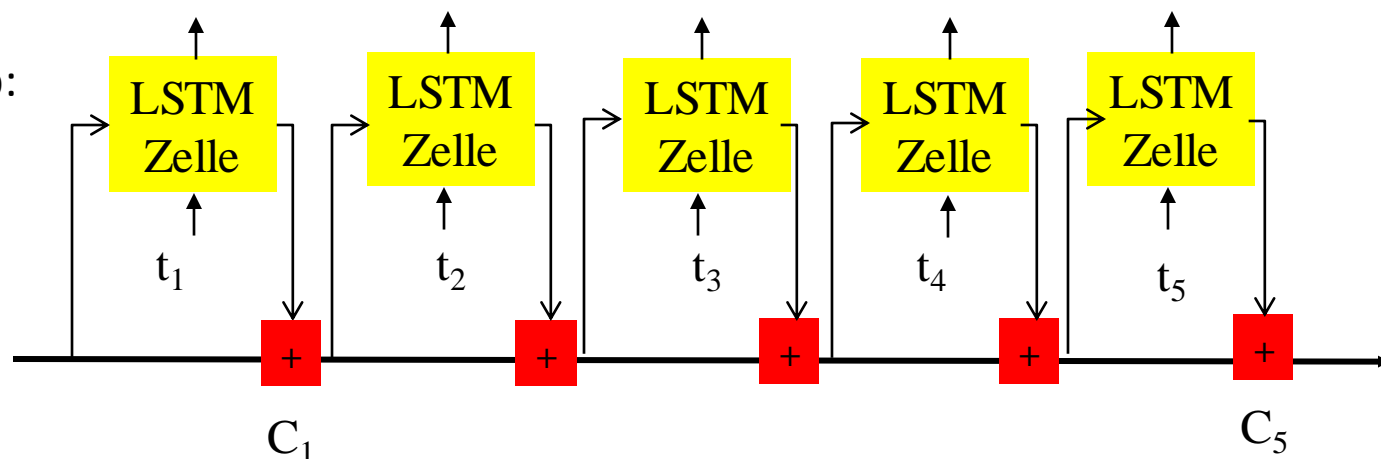
Speicher erforderlich:

- Speicherfähigkeit eines RNNs ist flüchtig: Bei jedem Zeitschritt werden alle Gewichte aller vorhergegangenen Zeitschritte automatisch überschrieben
- Vanishing Gradient Problem bewirkt, dass Gewichte zurückliegender Zeitschritte nur bedingt Einfluss auf neu berechnete Gewichte nehmen können.
- Benötigt wird eine Art Speicher, der die in den einzelnen Zeitschritten gespeicherte Information sichert.
- Ändern Anwendung der Gradienten von Multiplikation zu Addition zwischen den Zeitschritten

Lösung für das Vanishing Gradient Problem in RNNs: LSTM (Long Short Term Memory) Netze

- LSTM wurden 1997 von [Sepp Hochreiter](#) und [Jürgen Schmidhuber](#) publiziert und später von [Felix Gers](#) noch verbessert.
- LSTMs sind eine Spezialform der RNN Architektur, die das Vanishing Gradient Problem löst.
- Die spezielle Architektur der LSTMs ermöglicht es, komplexe Abhängigkeiten, die sich über sehr lange Zeitintervalle erstrecken, zu lernen.
- Es wird eine LSTM-Zelle eingeführt, die einen Zellzustandsvektor C_t bearbeitet.

• Prinzip:





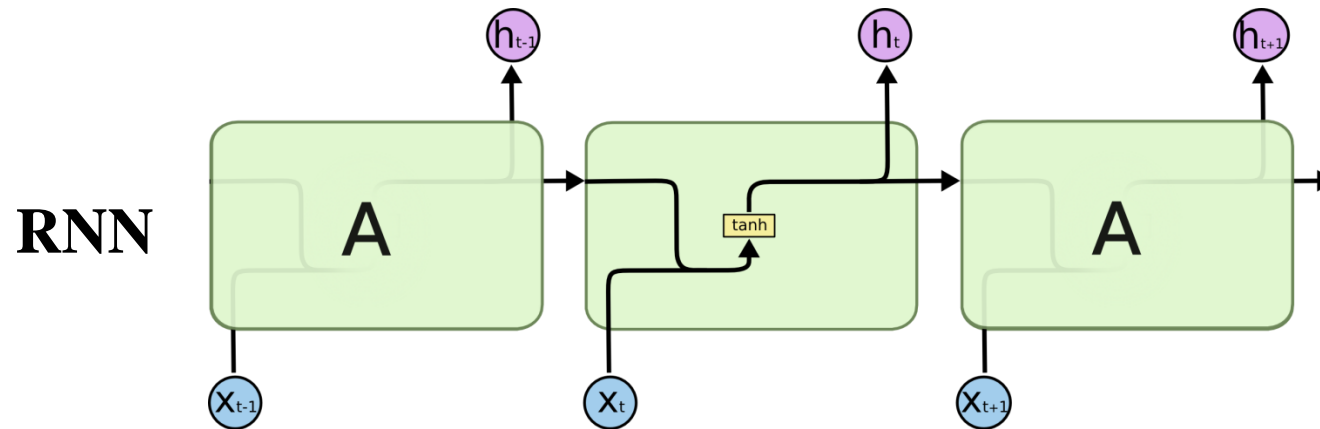
Lösung für das Vanishing Gradient Problem in RNNs: LSTM (Long Short Term Memory) Netze

Die **LSTM-Zelle** ist eine speziell entworfene Logikeinheit, die dazu beiträgt, das Problem des verschwindenden Gradienten ausreichend zu reduzieren, um rekurrente neuronale Netzwerke für Langzeitspeicheraufgaben einsetzbar zu machen

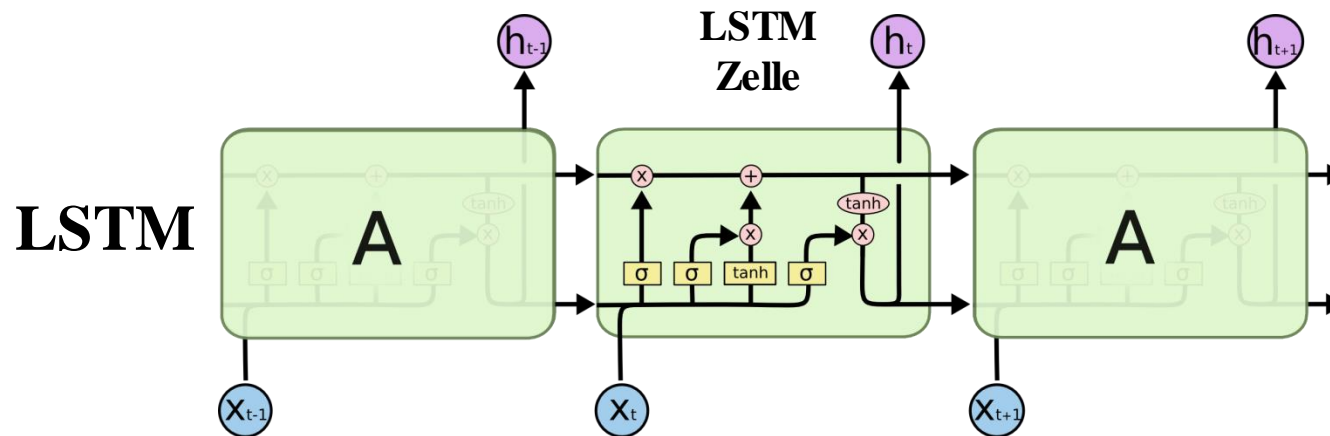
- **LSTM-Zellen** speichern Information, die in den einzelnen Zeitschritten vorhanden ist, in anderer Form → verwenden **Zellzustand**, der diese Information vor rekursiven Multiplikationen bei Backpropagation schützt.
- **LSTM-Zelle** kann selbstständig entscheiden, in welchem Maß zurückliegende Information (Kontext) zum aktuellen Zeitpunkt noch wichtig ist oder nicht.
- **LSTM-Zelle** ändert Zellzustand mit sogenannten **Zusatzgattern** und bestimmt neuen Output.



Standard RNN versus LSTM Architecture

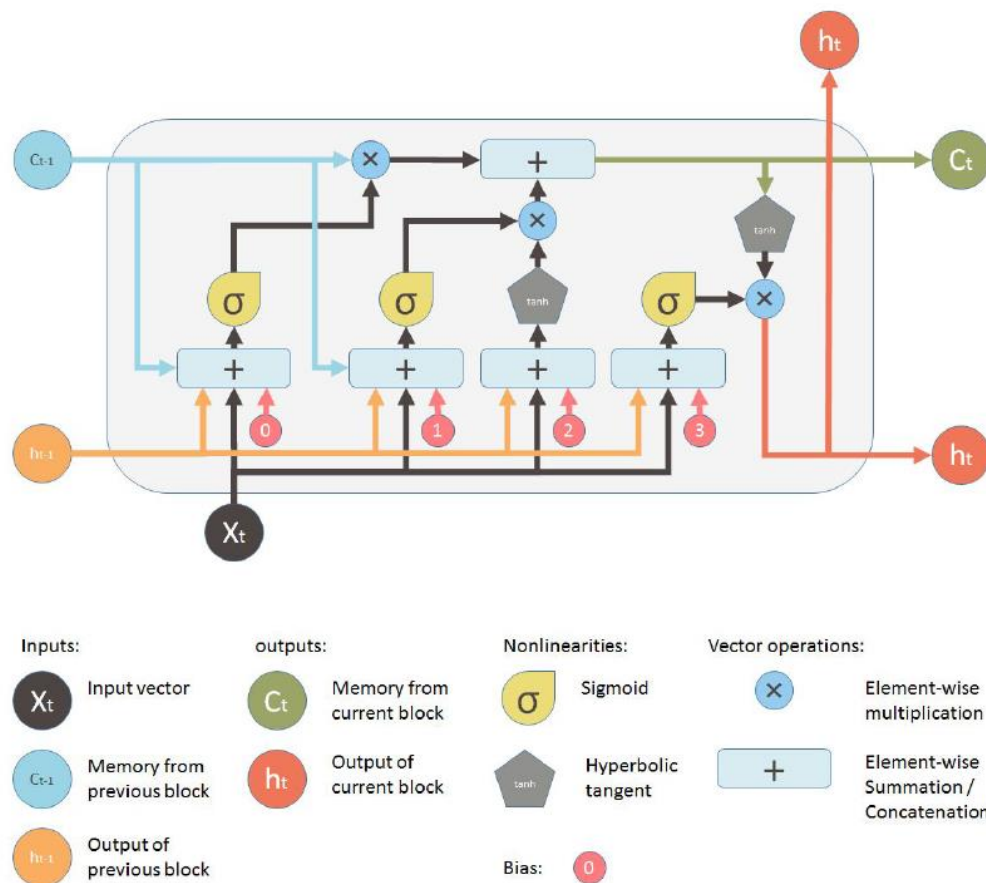


Repeating module in a standard RNN contains a single layer



Repeating module in an LSTM is a memory cell

LSTM Memory Cell



Zentrale Idee:

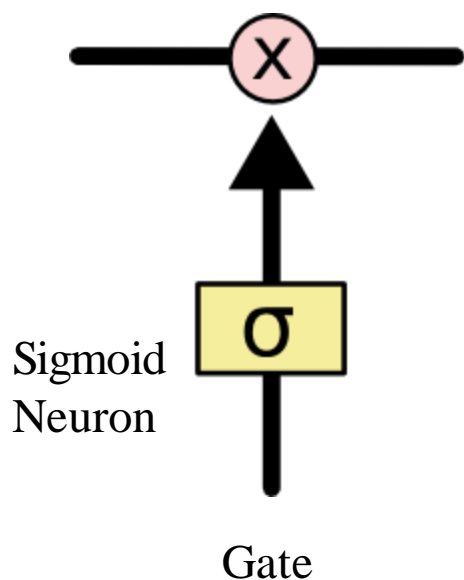
- Eine Speicherzelle (**memory cell** oder **block**), die ihren Zustand im Zeitablauf selbst verwalten kann.
- Sie enthält einen expliziten Speicher, dem sogenannten **Zellzustandsvektor**.
- Dazu kommen noch verschiedene **Gatter**, die regeln, welche Information in und aus dem Speicher fließt.

Der Schlüssel zu LSTMs ist der **Zellzustandsvektor** C_t , die horizontale Linie, die am oberen Rand des Diagramms verläuft.

LSTM Gates

Die LSTM Zelle hat die Fähigkeit, Informationen aus dem **Zellzustandsvektor** zu entfernen oder hinzuzufügen, was durch sogenannte als **Gates** bezeichnete Strukturen reguliert wird.

Gates bestehen aus **sigmoid** bzw. **tanh** Neuronen und einer punkweisen Multiplikationsoperation.



Das Sigmoid-Neuron gibt Zahlen zwischen Null und Eins aus und beschreibt, wie viel von jeder Komponente durchgelassen werden sollte.

Ein Wert von Null bedeutet "**nichts durchlassen**", während ein Wert von eins "**alles durchlassen**" bedeutet.

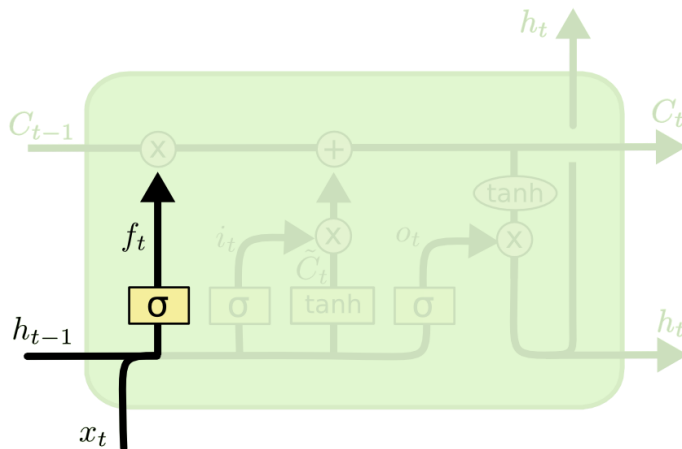
Ein LSTM hat drei (vier) dieser Gates, um den Zellzustand zu schützen und zu steuern.

LSTM Forget Gate

Der erste Schritt im LSTM ist zu entscheiden, welche Informationen aus dem hidden state h_{t-1} der vorhergehenden Zelle und dem aktuellen Input x_t weiter verwendet werden sollen.

Diese Entscheidung wird durch das "Forget Gate" vorgenommen. Es betrachtet h_{t-1} und x_t und gibt einen Vektor mit Zahlen zwischen 0 und 1 (sigmoid) aus für jede Zahl im Zellzustandsvektor C_{t-1} .

Eine 1 steht für "behalte diese vollständig", während eine 0 für "ganz loswerden" steht.



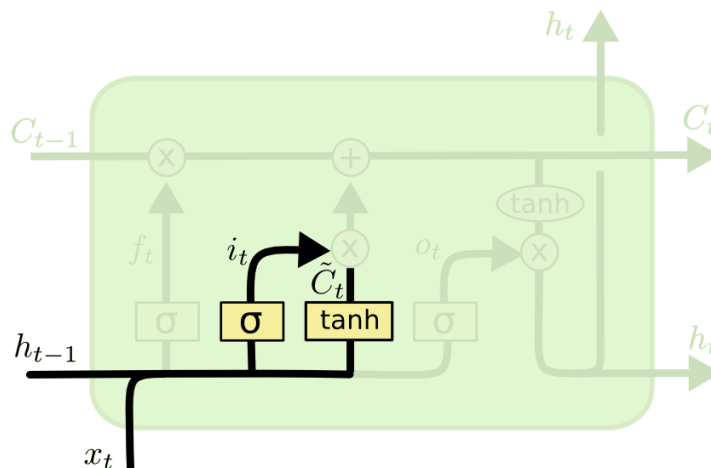
$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

LSTM Input Gate und Gate g?

Der nächste Schritt besteht darin, zu entscheiden, welche neuen Informationen im Zellzustand zu speichern sein werden.

g?: Dieses Gate hat keinen Namen. Es ist zum Teil verantwortlich für das Hinzufügen von Information zum Zellzustand. Durch das Anwenden eines **tanh**-Neurons auf h_{t-1} und x_t wird der Ergebnisvektor auf Zahlen zwischen $[-1 \ 1]$ gemappt. Es liefert den Update- Vektor \tilde{C}_t .

Input Gate: Dieses Gate beeinflusst, wieviel von \tilde{C}_t zum Zellzustand addiert werden soll. Die Ausgabe des Input Gates ist zwischen $[0 \ 1]$ (sigmoid) und die Ausgabe von **g?** ist zwischen $[-1 \ 1]$ (tanh). Nach der Multiplikation beider Ausgabevektoren wird der Zellzustand durch Addition dieses Vektors vergrößert oder verkleinert.



$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

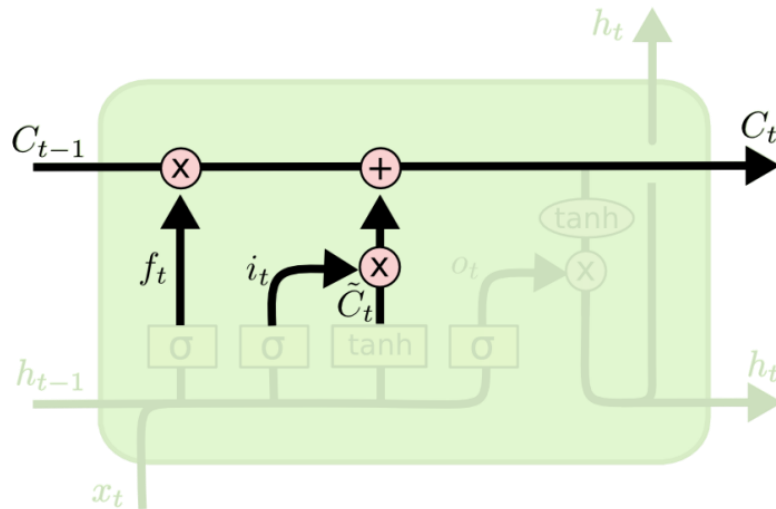
LSTM Forget Gate und Input Gate

Ergebnis:

Der alte Zellzustand C_{t-1} wurde mit f_t multipliziert und die Informationen, die vom Forget Gate festgelegt wurden wurden entfernt.

Dann wird noch $i_t * \tilde{C}_t$ durch Addition hinzugefügt.

Dies liefert den neuen Zellzustand C_t .



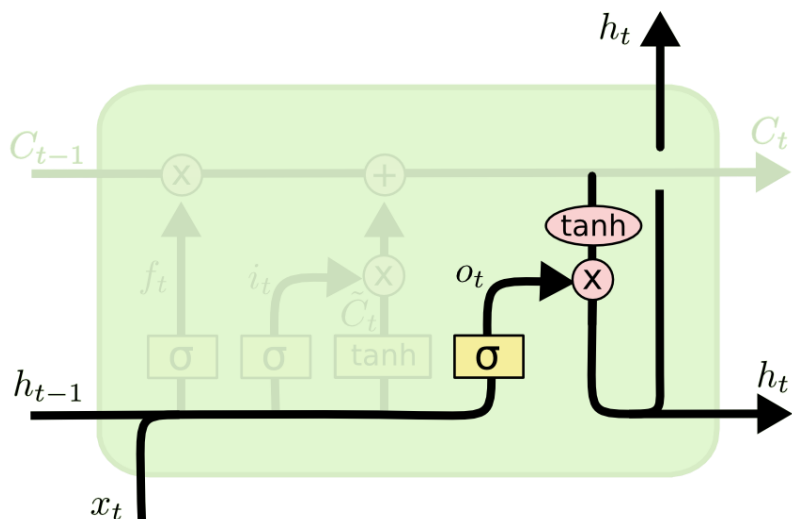
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM Output Gate

Schließlich wird entschieden, was der neue hidden State \mathbf{h}_t der Zelle ist.

Diese Ausgabe basiert auf dem hidden State der letzten Zelle \mathbf{h}_{t-1} und dem aktuellen Input \mathbf{x}_t , die wiederum durch ein sigmoid-Neuron auf Werte zwischen [0 1] abgebildet werden.

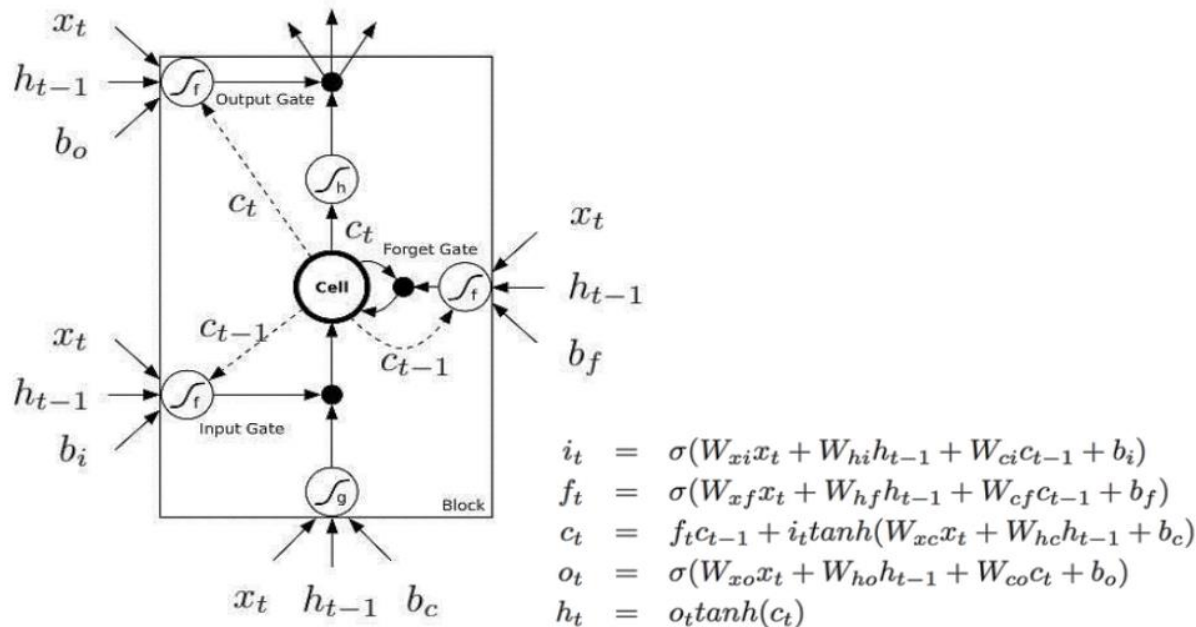
Dann wird der aktuelle Zellzustand \mathbf{C}_t durch **tanh** (Werte zwischen -1 und 1) und mit dem Ausgang des **Sigmoid**-Neurons verknüpft, so dass nur die Teile ausgegeben werden, für die wir uns entschieden haben. Dies ist der neu hidden State \mathbf{h}_t der Zelle.



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Zusammenfassung LSTM



Eine LSTM-Speicherzelle besteht aus einem Zellzustand, der als Gedächtnis fungiert und drei Gates, die den Zellzustand beschützen und kontrollieren.

Jedes Gate arbeitet mit einem Sigmoid-Neuron, das einen Wertebereich zwischen 0 und 1 ausgibt und damit die Intensität der Aktion bestimmt.

Das Forget Gate ist für das Löschen zuständig, das Eingangsgate übernimmt die Aktion das Neu-Merkens, indem es neue Informationen in den Zellzustand speichert und das Ausgangsgate bestimmt die Informationen, die ausgegeben werden.

RNNs überschreiben den hidden state

LSTMs fügen etwas zum hidden state hinzu

Zusammenfassung LSTM

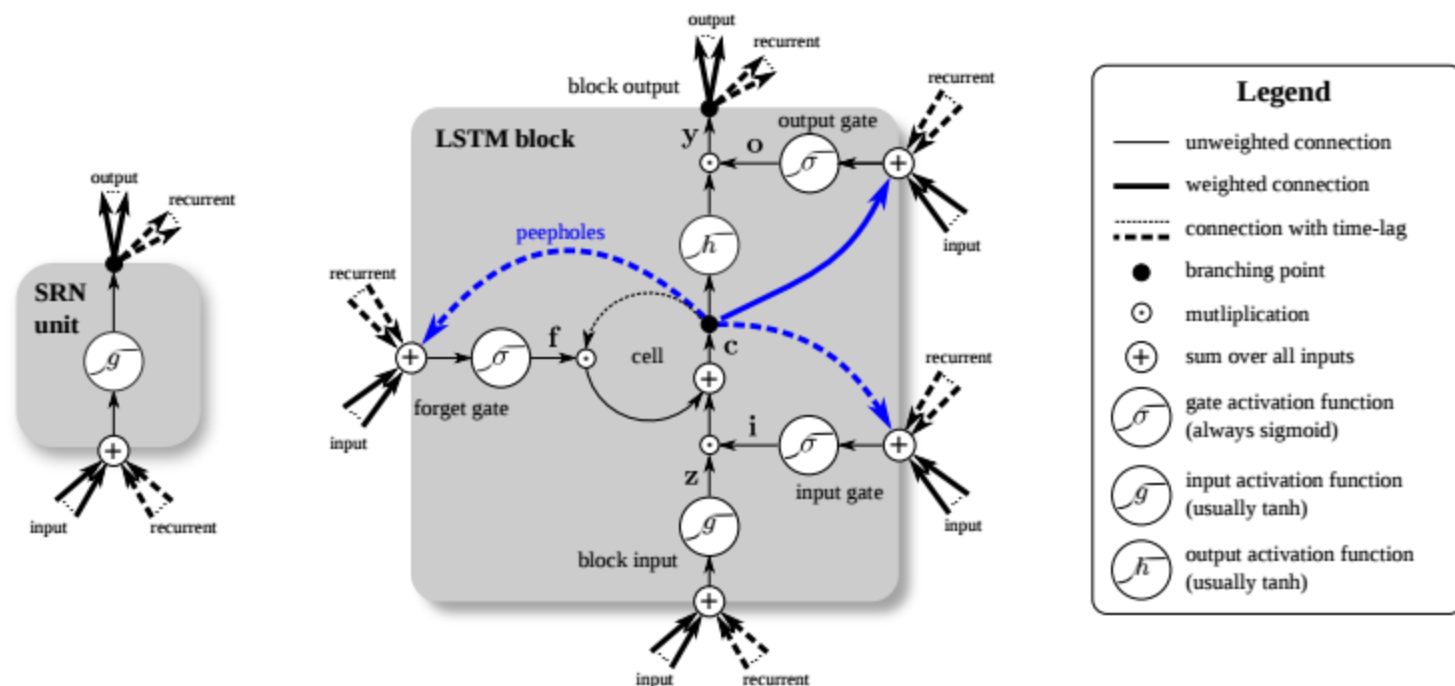


Figure 1. Detailed schematic of the Simple Recurrent Network (SRN) unit (left) and a Long Short-Term Memory block (right) as used in the hidden layers of a recurrent neural network.



Einsatzgebiete von NN

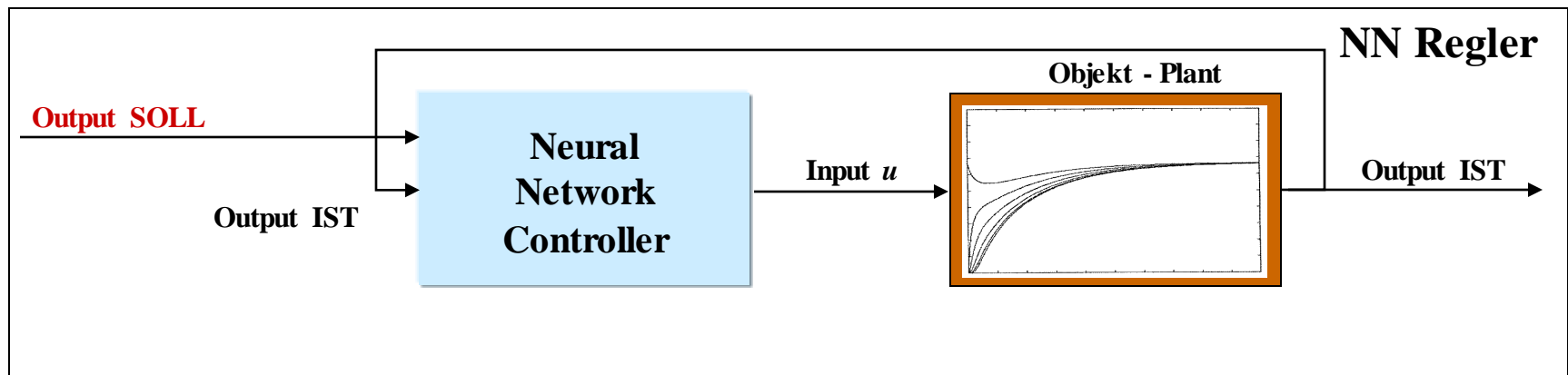
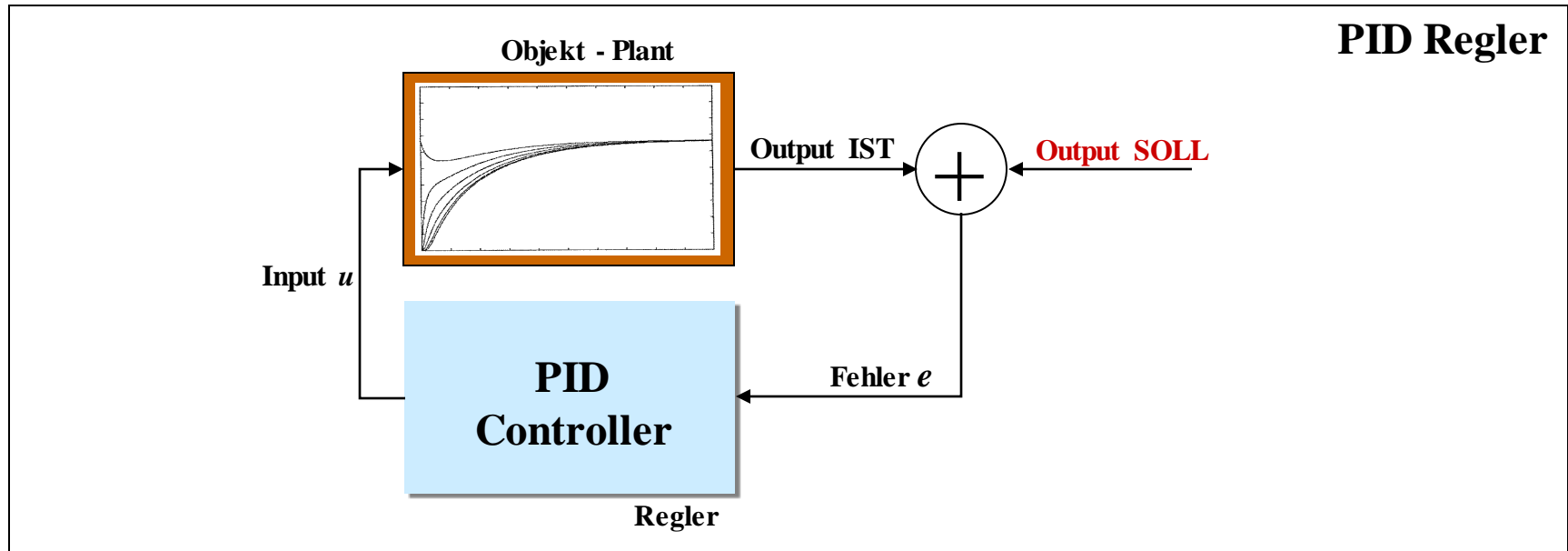
Neurocomputing
Teil 3

Einsatzgebiete von NN

- Datamining
- Business Intelligence
- Mustererkennung (natürliche Sprache, Schriftzeichen, Semantik)
- Datenkompression
- Lösungen von kombinatorischen Problemen
- Regelungs- und Steuerungssysteme
- Finanz und Ökonomie
 - Finanz-Analyse -- Stock Prediktion
 - Signaturen-Analyse -- Bank nutzen NNs für Vergleichs Signaturen mit Muster.
 - Direktes Marketing -- NN beobachtet Ergebnisse von Mailing-Test und determiniert die besten Bereiche.
- Medizinische Informations- und Expertensysteme

Einsatzgebiete von NN

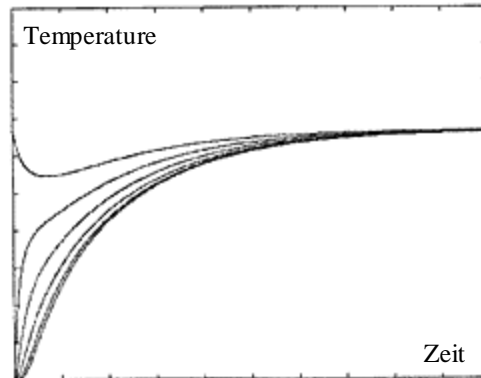
Beispiel: Control System



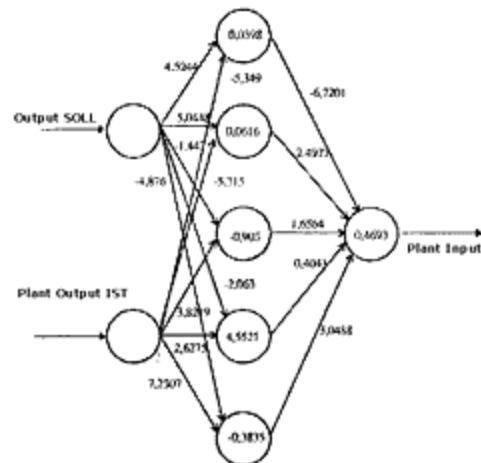
Einsatzgebiete von NN

Beispiel: Control System

Plant – Steuerungsobjekt:
Hochofen

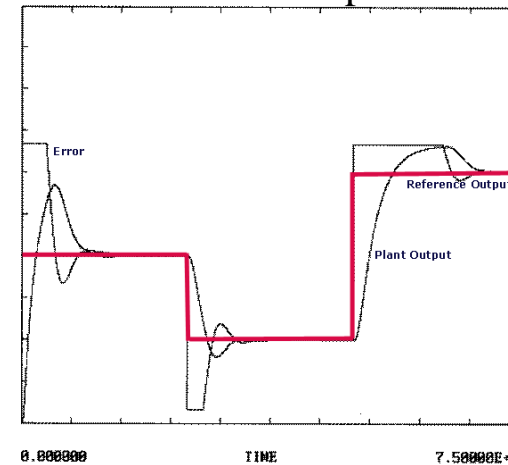


Neural Network Controller



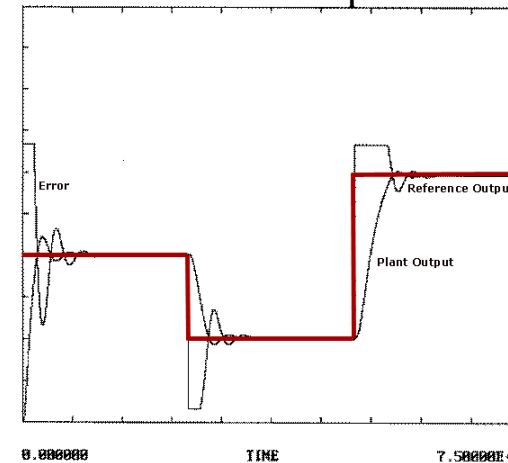
**PID
Controller**

Plant Output/Error



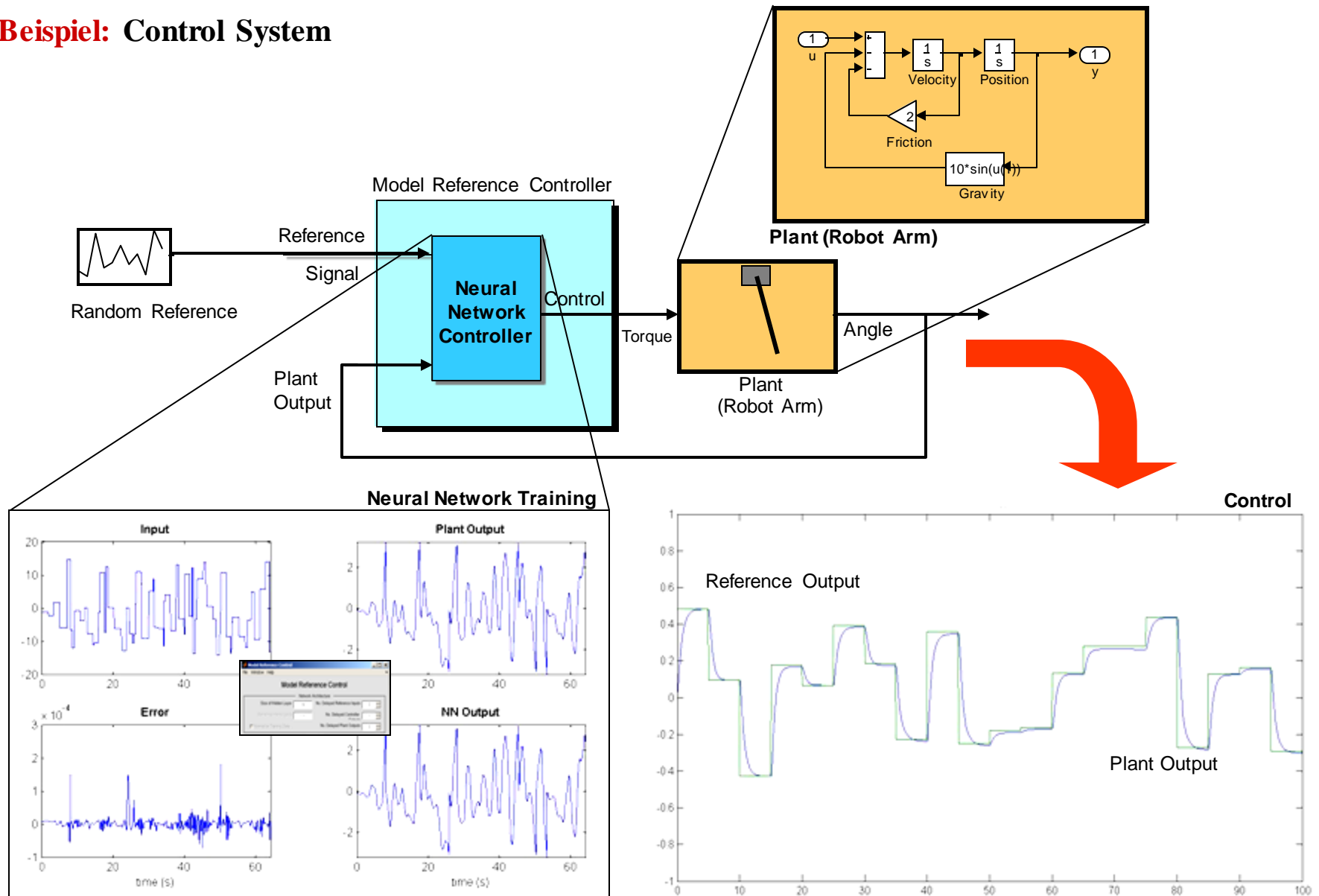
**Neural
Controller**

Plant Output/Error



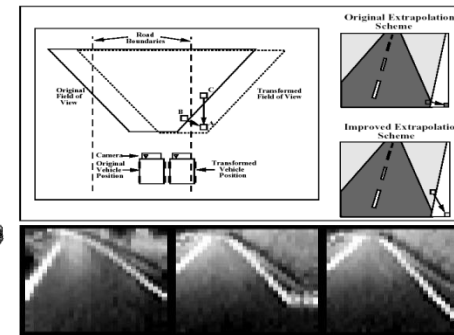
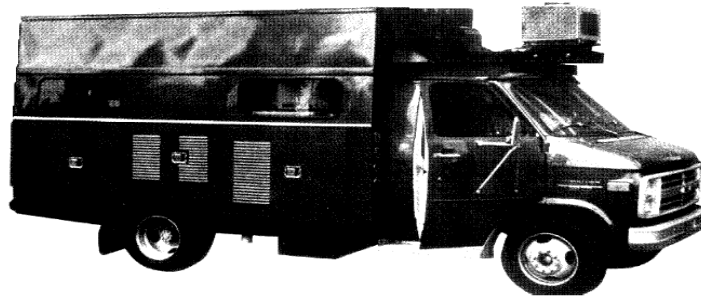
Einsatzgebiete von NN

Beispiel: Control System

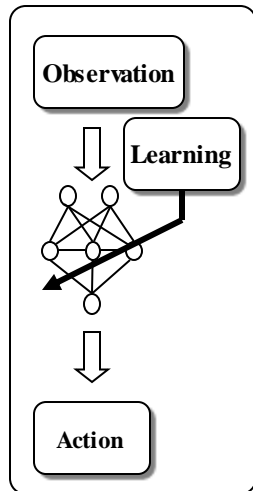


Einsatzgebiete von NN

Beispiel: Neural Network Vision based Control of Mobile Vehicles



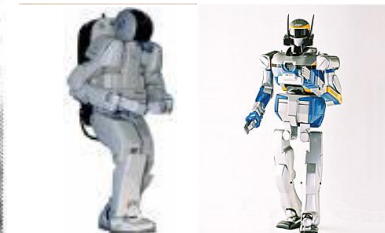
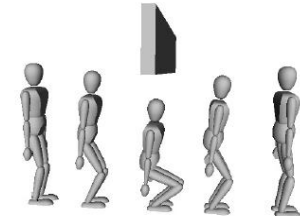
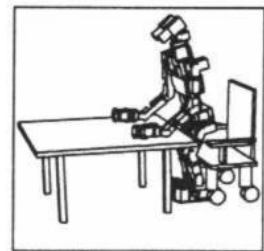
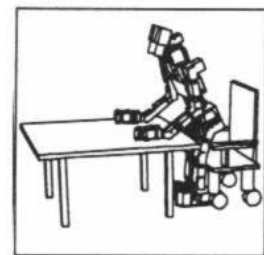
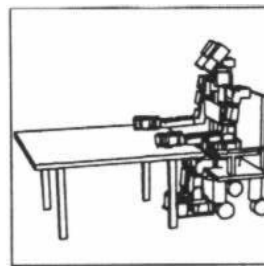
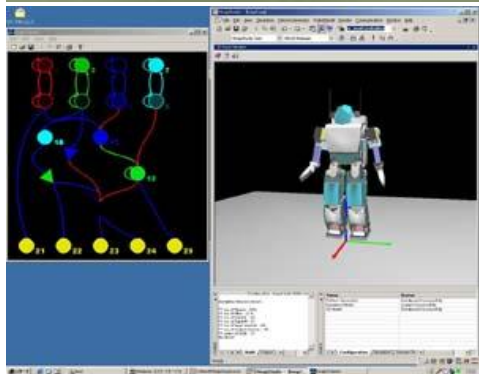
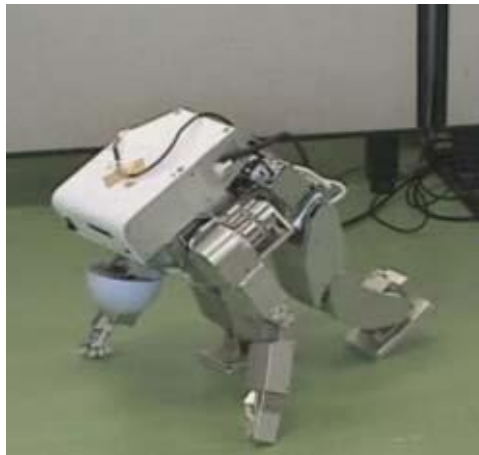
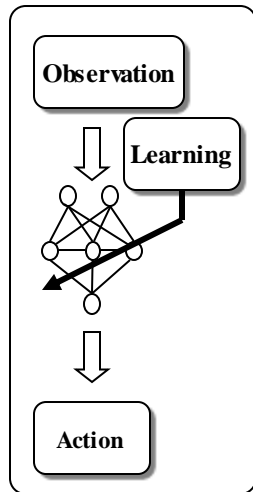
The basic network architecture employed in the **ALVINN system** is a single hidden layer feedforward neural network. The input layer now consists of a single 30x32 unit-retina onto which a sensor image from either a video camera or a scanning laser rangender is projected. Each of the 960 input units is fully connected to the hidden layer of 4 units, which is in turn fully connected to the output layer. The 30 unit output layer is a linear representation of the currently appropriate steering direction which may serve to keep the vehicle on the road or to prevent it from colliding with nearby obstacles. (Carnegie Mellon University)



The **Mars Exploration Rovers will** use basic forms of behavior-based control to perform certain tasks autonomously on the surface of Mars. Rovers can be trained by humans, in a laboratory environment, how to recognize and move through difficult terrain. The learned skill can then be applied when deployed at a distant site, such as the marsian surface. (NASA Jet Propulsion Laboratory)

Einsatzgebiete von NN

Beispiel: Humanoid Movement Generation System

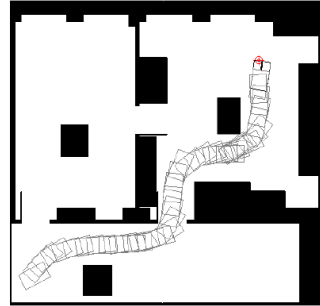
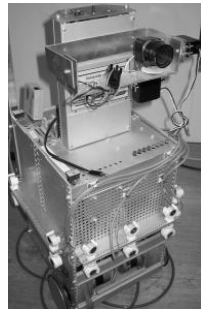


Neural oscillator based Humanoid Movement Generation System, which enables humanoid robots to learn a wide range of movements (Fujitsu Research Lab, Honda P3 Humanoid Robot, Hansen Robots Texas, Stanford Uni).



Einsatzgebiete von NN

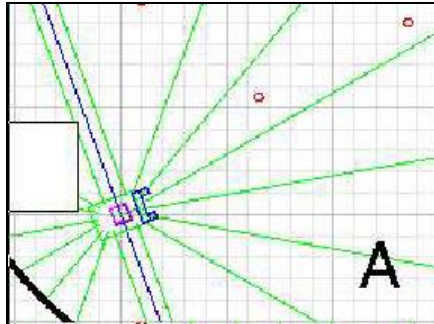
Beispiel: Neural Network Controllers for Autonomous Robots



Observation

Learning

Action



The method of the construction of a collision-free path for moving robot among obstacles is based usually on two neural networks. The first neural network is used to determine the “free” space using ultrasound range finder data. The second neural network “finds” a safe direction for the next robot section of the path in the workspace while avoiding the nearest obstacles (Univ. of Mass., University of Essex, SAS, Stanford University).

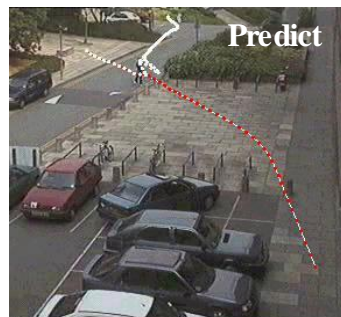
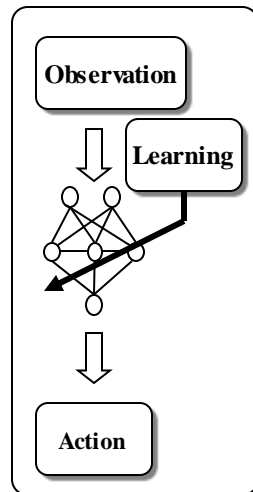


Here are two EvBots trying to find their way (Center for Robotics and Intelligent Machines (CRIM), North Carolina State University)



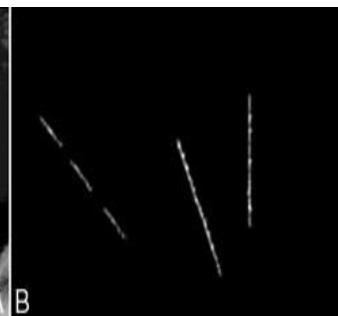
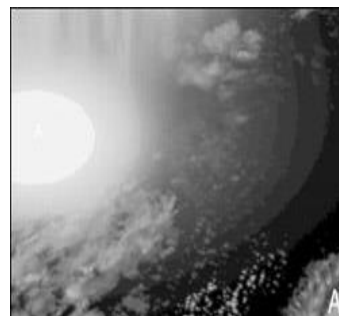
Einsatzgebiete von NN

Beispiel: Image Processing



Learning the Distribution of Object Trajectories:

The NN learnt in an unsupervised manner by tracking objects over long image sequences. NN Models of the trajectories of pedestrians have been generated and used to assess the typicality of new trajectories, predict future object trajectories (University of Leeds)



Detection and Tracking of Moving Targets:

Raw input backgrounds with weak targets included. Detected target sequence at the ANN processing output, post-detection tracking not included (Defense Group Incorporated).

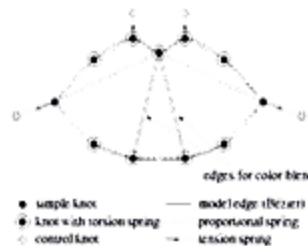
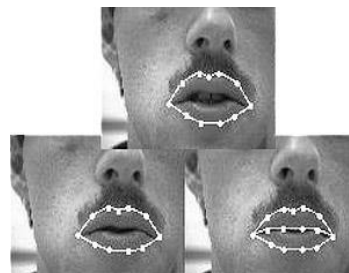
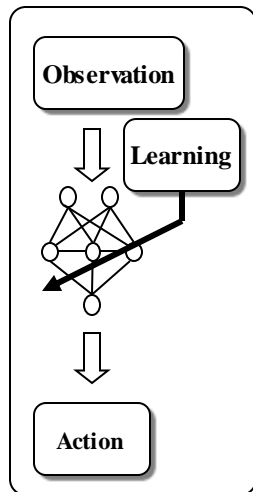


Image Processing - Speechreading (Lipreading):

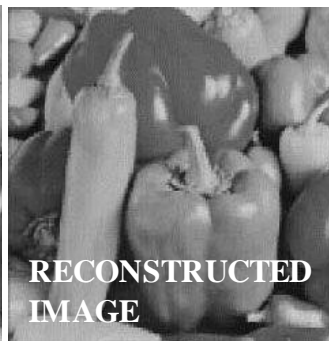
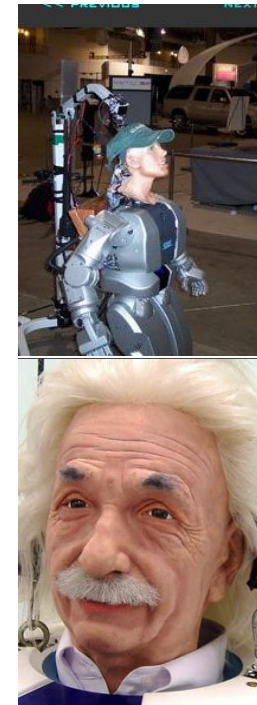
Automatic speechreading is based on a robust lip image analysis. The system allows for realtime tracking and storage of the lip region. A neural classifier detects visibility of teeth edges and other attributes. The resulting parameters may be used for speech recognition tasks.(Universität Stuttgart)

Einsatzgebiete von NN

Beispiel: Neural Network based Facial Animation-Control and Data Compression



Facial animations created using hierarchical B-spline as the underlying surface representation. Neural networks could be used for learning of each variation in the face expressions for animated sequences (University of British Columbia, Hanson Robotics Texas).



A neural net suitable for solving the **image compression problem** (bottleneck type network) consists of an input layer and an output layer of equal sizes, with an intermediate layer of smaller size in-between. The ratio of the size of the input layer to the size of the intermediate layer is the compression ratio



Literatur:

Neurocomputing:

1. *Neural Networks: A Comprehensive Foundation*, S. Haykin, Prentice Hall, 1999.
2. *An Introduction to Neural Networks*, K. Gurney, UCL Press, 1997
3. *The Essence of Neural Networks*, R. Callan, Prentice Hall, Europe, 1999
4. *Fundamentals of Neural Networks*, L. Fausett, Prentice Hall, 1994
5. *Introduction to Neural Networks*, R. Beale & T. Jackson IOP Publishing, 1990
6. *An Introduction to the Theory of Neural Computation*, J. Hertz, A. Krogh & R.G. Palmer, Addison Wesley, 1991
7. *Parallel Distributed Processing: Volumes 1 and 2*; D.E. Rumelhart, J.L. McClelland, et al., MIT Press, 1986, (The original neural networks bible).
8. *The Computational Brain*; P.S. Churchland & T.J. Sejnowski, MIT Press, 1994
9. *Principles of Neurocomputing for Science and Engineering*; F.M. Ham & I. Kostanic, McGraw Hill, 2001
10. *Neural Networks for Pattern Recognition*, C. Bishop, Clarendon Press, Oxford, 1995
11. *Praktikum Neuronale Netze*, H. Braun & R. Malaka, Springer, 1997

Neurocomputing Applications:

1. *Fuzzy and Neural Systems in Medicine*, H. Teodorescu & A. Kandel, CRC Press 1999
2. *Soft Computing in Financial Engineering*, R. Ribeiro & R. Yager (ed.), Springer, 1999
3. *Intelligent Robotic Systems*, W. Jacak, Kluwer Academic, NY, 2000
4. *Soft Computing and Intelligent Systems*, M. Gupta, Academic Press, 2000
5. *Explanation based Neural Network Learning*, S. Thurn, Kluwer Academic, 1996

Journal: *IEEE Transaction on Neural Networks*