# Chapter 11:

# Recurrent Neuro Fuzzy Systems

# Motivation

Neuro-fuzzy models are usually used if

- Vague knowledge can be included into the solution (i.e. we know something about our data or a possible solution)
- The solution should be interpretable in form of rules (i.e. we want to learn something about our data/problem)
- From an applicational point of view the solution should be easy to implement, to use and to understand.
- Interpretation is more important than performance

# Motivation

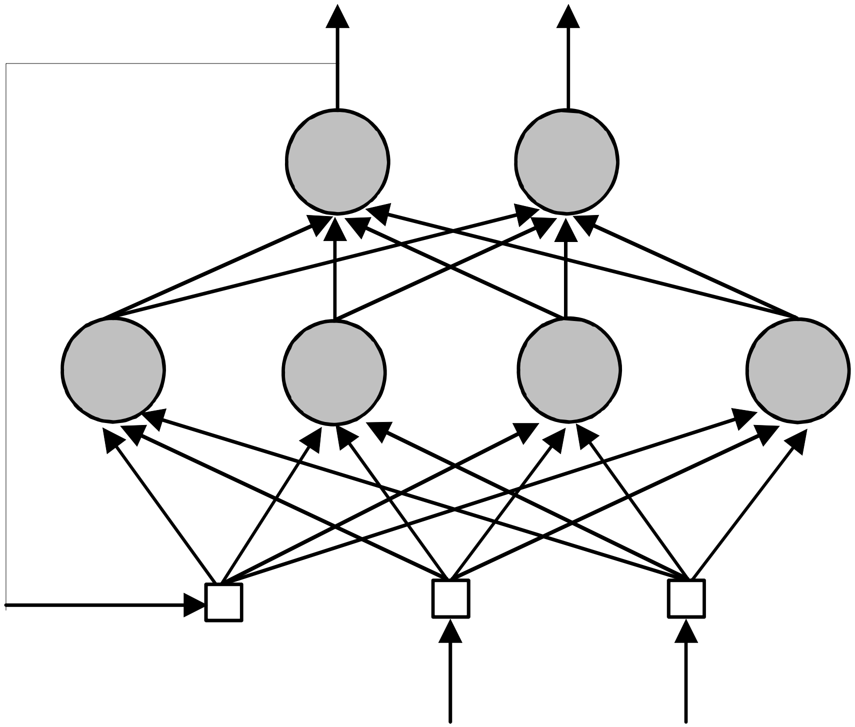
Conventional (feedforward) neuro-fuzzy models less appropriate for control / analysis of higher order dynamic systems or time series data:

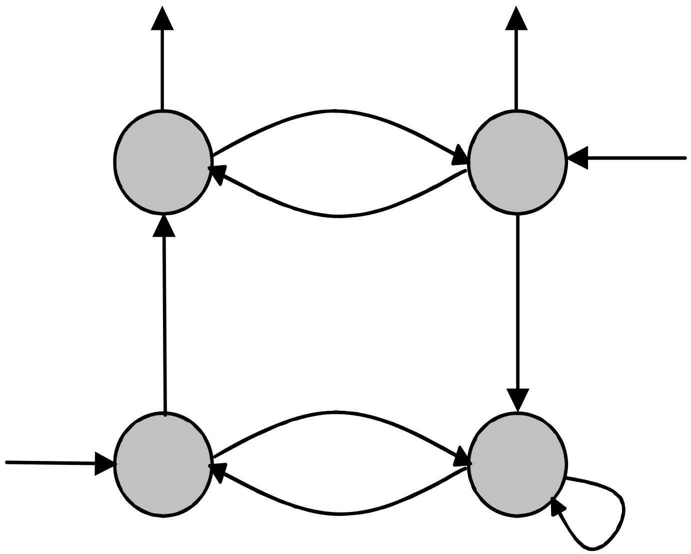
- obtained systems usually very complex (i.e. number of inputs and rules very high)
- obtained systems are usually hard to interpret
- performance decreases
- initialization by use of linguistic rules usually impossible

# Recurrent Neural Networks

- Backward connections are used to model time delayed feedback (‚information of the past‘)
- Mathematical complexity increases: RCNN‘s are universal approximators and can be used to model systems of higher order ordinary differential equations [Funashi/Nakamura, 1993]
- Different architectures have been proposed, e.g.:
  - Hopfield networks [Hopfield, 1982]
  - Partially recurrent networks (e.g. recurrent multilayer perceptron [Puskorius/Feldkamp, 1994])
  - Fully recurrent networks

# Recurrent Neural Networks (Examples)



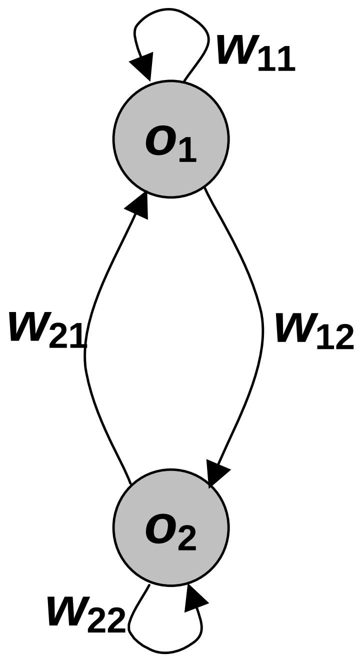


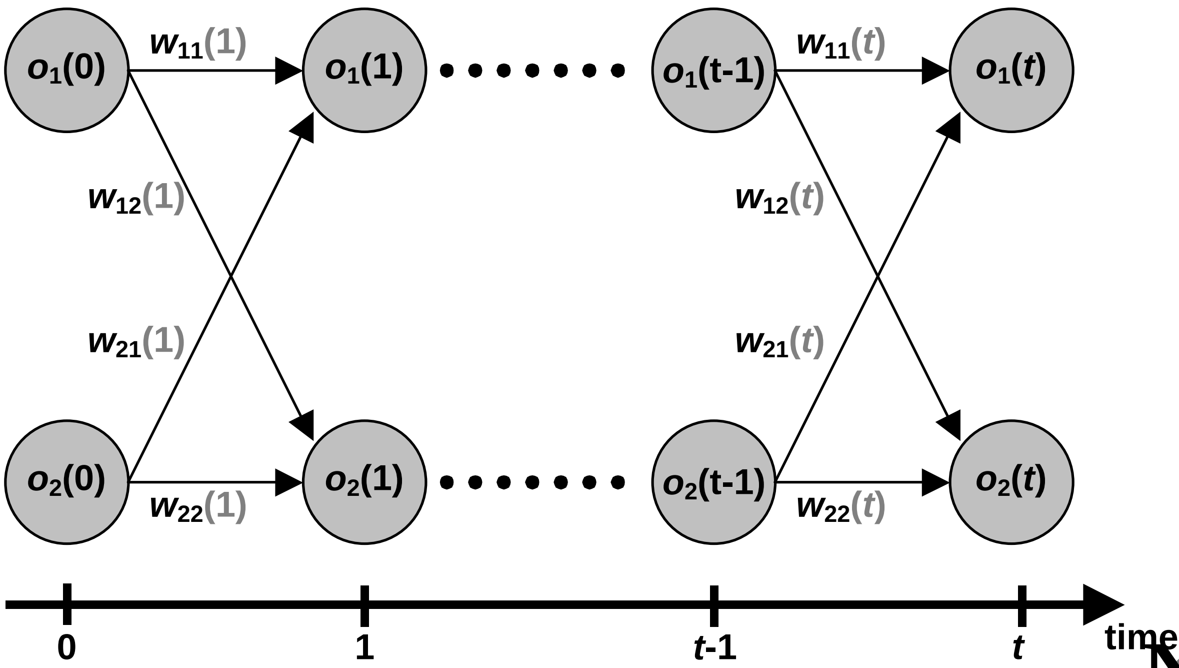
$$o_j(t) = f(net_j(t)) = f(\sum_i o_i(t-\Delta t)w_{ij} + ext_j(t-\Delta t))$$ (difference equation)

$$\frac{do_j}{dt} = -o_j + f(\sum_i o_i w_{ij} + ext_j)$$ (differential equation)

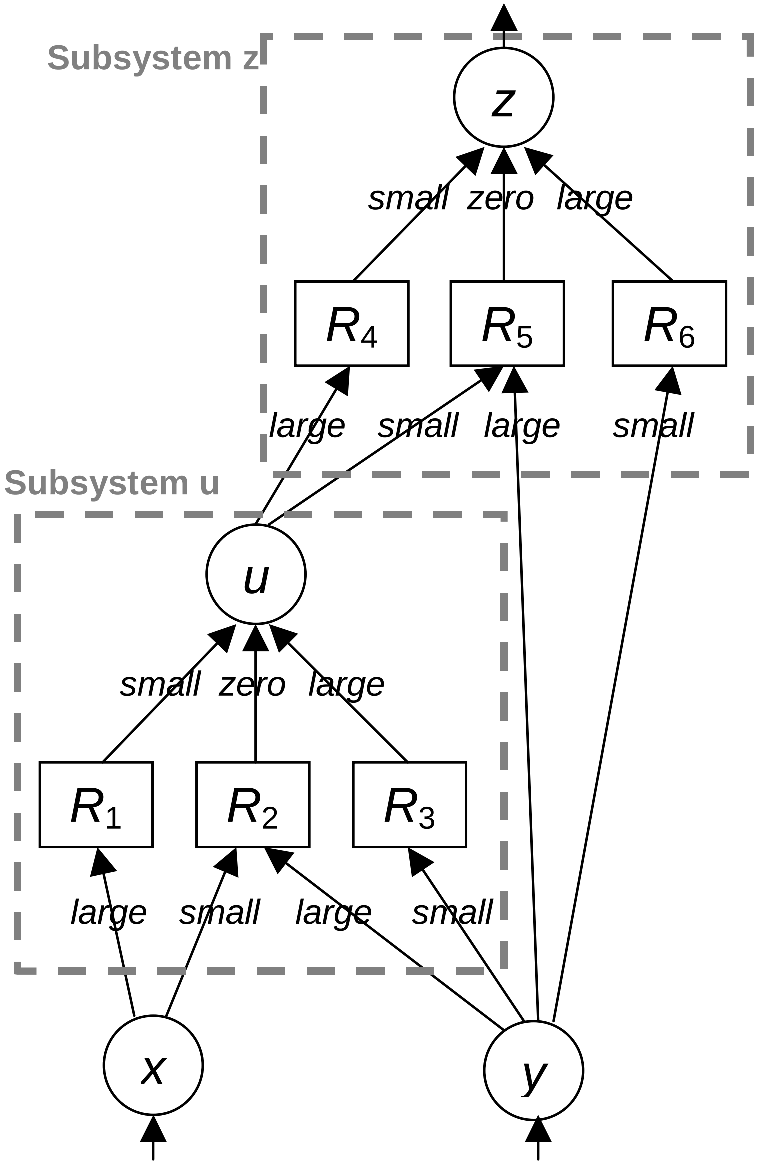
# Recurrent Neural Networks

- Basic learning methods:
  - Backpropagation through time [Rumelhart et al., 1986]
  - Real time recurrent learning [Williams and Zipser, 1989]



# Fuzzy System (hierarchical)

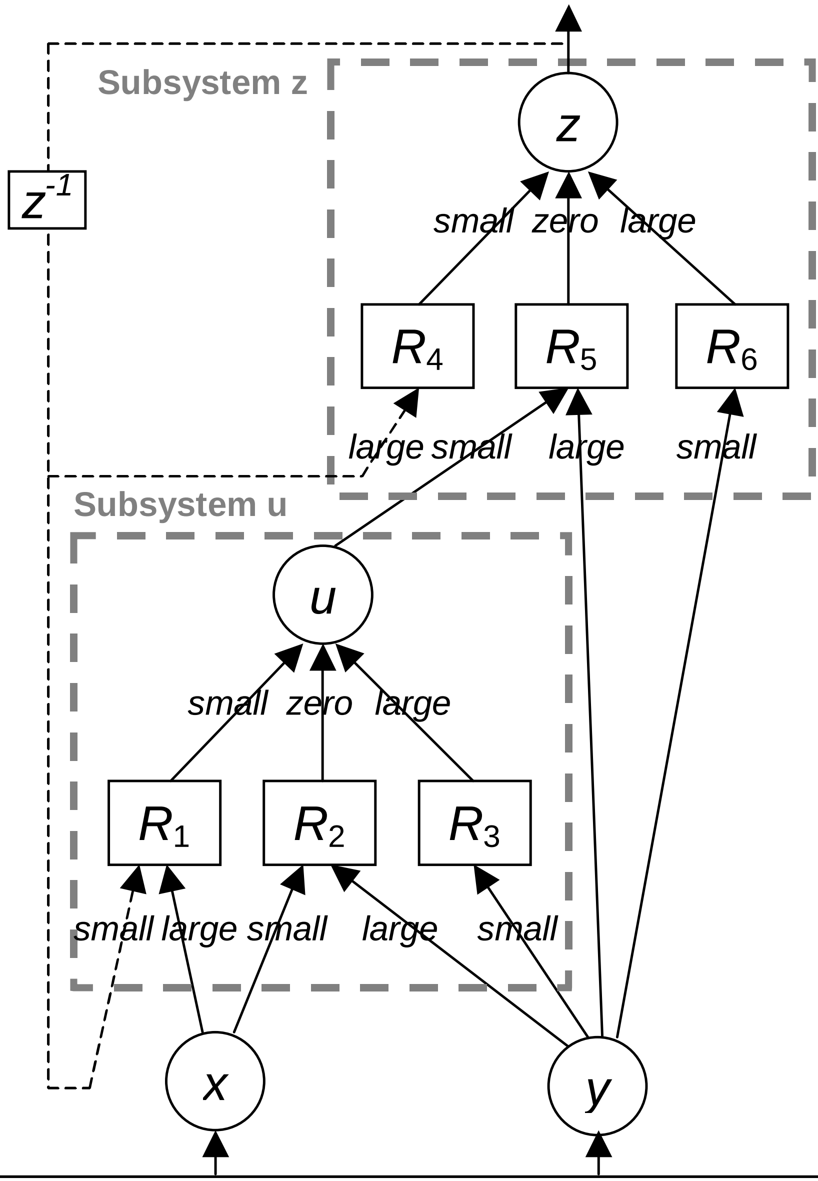



Input variables: $x$, $y$
Output variables: $z$
Inner variables: $u$

$R_1$: **If** $x$ is *large*
**then** $u$ is *small*
$R_2$: **If** $x$ is *small* **and** $y$ is *large*
**then** $u$ is *zero*
$R_3$: **If** $y$ is *small*
**then** $u$ is *large*

$R_4$: **If** $u$ is *large*
**then** $z$ is *small*
$R_5$: **If** $u$ is *small* **and** $y$ is *large*
**then** $z$ is *zero*
$R_6$: **If** $y$ is *small*
**then** $z$ is *large*

# Fuzzy System (hierarchical and recurrent)




Input variables: $x$, $y$
Output variables: $z$
Inner variables: $u$

$R_1$: **If** $x(t)$ is *large* **and** $z(t\text{-}1)$ is *large*
**then** $u(t)$ is *small*
$R_2$: **If** $x(t)$ is *small* **and** $y(t)$ is *large*
**then** $u(t)$ is *zero*
$R_3$: **If** $y(t)$ is *small*
**then** $u(t)$ is *large*

$R_4$: **If** $z(t\text{-}1)$ is *large*
**then** $z(t)$ is *small*
$R_5$: **If** $u(t)$ is *small* **and** $y(t)$ is *large*
**then** $z(t)$ is *zero*
$R_6$: **If** $y(t)$ is *small*
**then** $z(t)$ is *large*

# Fuzzy System (recurrent)

A recurrent fuzzy system of the form

$$R_r : \textbf{If } y(t\text{-}1) \text{ is } \mu_r \textbf{ then } y(t) \text{ is } \nu_r$$

can be used to approximate an initial value problem of the form

$$\dot{y} = f(y), \quad y(t_0) = c$$

$$\frac{\Delta y}{\Delta t} = f(y), \quad y(t_0) = c$$ (difference quotient)

$$y_{i+1} = y_i + \Delta t \cdot f(y_i), \quad y_0 = c$$ (iterative solution)

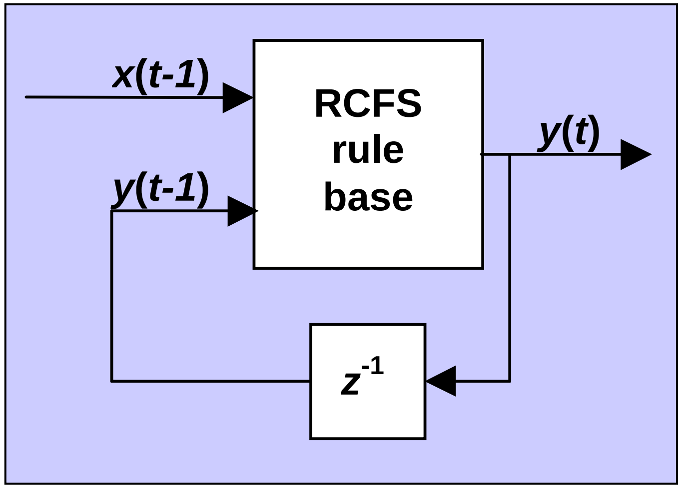
$$S(y_i) = y_i + h \cdot g(y_i)$$ ($S$ defined by fuzzy system)

$$y_{i+1} = y_i + \Delta t \cdot f(y_i) = y_i + h \cdot g(y_i) = S(y_i)$$

# Recurrent Fuzzy Systems (First order)

A **first order recurrent fuzzy system** consist of rules of the form:

**If** $x(t\text{-}1)$ is $A$ **and** $y(t\text{-}1)$ is $B'$ **then** $y(t)$ is $B$




System state diagram

| | | | |
|---|---|---|---|
| **If** | $y(t\text{-}1)$ is *negative* | **and** $x(t\text{-}1)$ is *negative* | **then** $y(t)$ is *negative* |
| **If** | $y(t\text{-}1)$ is *negative* | **and** $x(t\text{-}1)$ is *zero* | **then** $y(t)$ is *negative* |
| **If** | $y(t\text{-}1)$ is *negative* | **and** $x(t\text{-}1)$ is *positive* | **then** $y(t)$ is *zero* |
| **If** | $y(t\text{-}1)$ is *zero* | **and** $x(t\text{-}1)$ is *negative* | **then** $y(t)$ is *negative* |
| **If** | $y(t\text{-}1)$ is *zero* | **and** $x(t\text{-}1)$ is *zero* | **then** $y(t)$ is *zero* |
| **If** | $y(t\text{-}1)$ is *zero* | **and** $x(t\text{-}1)$ is *positive* | **then** $y(t)$ is *positive* |
| **If** | $y(t\text{-}1)$ is *positive* | **and** $x(t\text{-}1)$ is *negative* | **then** $y(t)$ is *zero* |
| **If** | $y(t\text{-}1)$ is *positive* | **and** $x(t\text{-}1)$ is *zero* | **then** $y(t)$ is *positive* |
| **If** | $y(t\text{-}1)$ is *positive* | **and** $x(t\text{-}1)$ is *positive* | **then** $y(t)$ is *positive* |

Rule base of a simple limited integrator

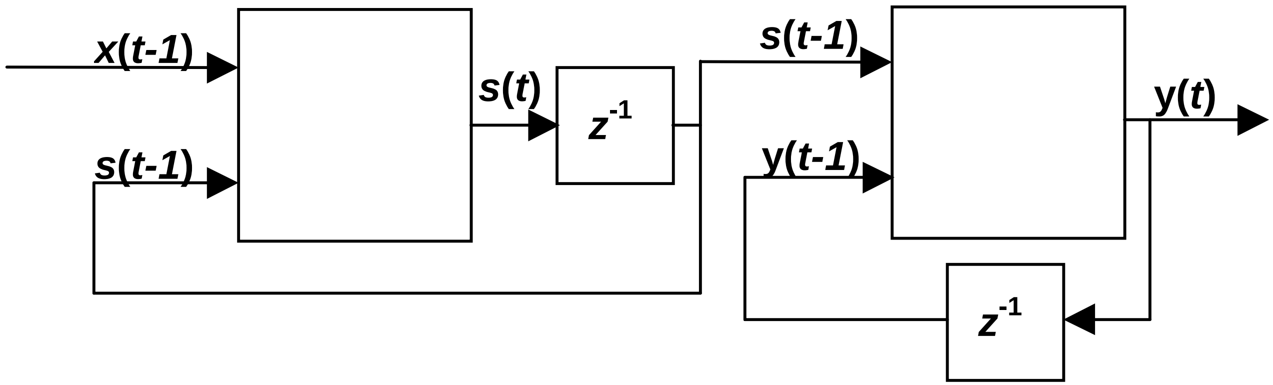
# Recurrent Fuzzy Systems (Higher order example)

Using rules of the form

**If** $x(t\text{-}1)$ is $A_i$ **and** $s(t\text{-}1)$ is $C_i$ **then** $s(t)$ is $C_i$'

**If** $y(t\text{-}1)$ is $B_i$ **and** $s(t\text{-}1)$ is $C_i$ **then** $y(t)$ is $B_i$'

we obtain the following system:

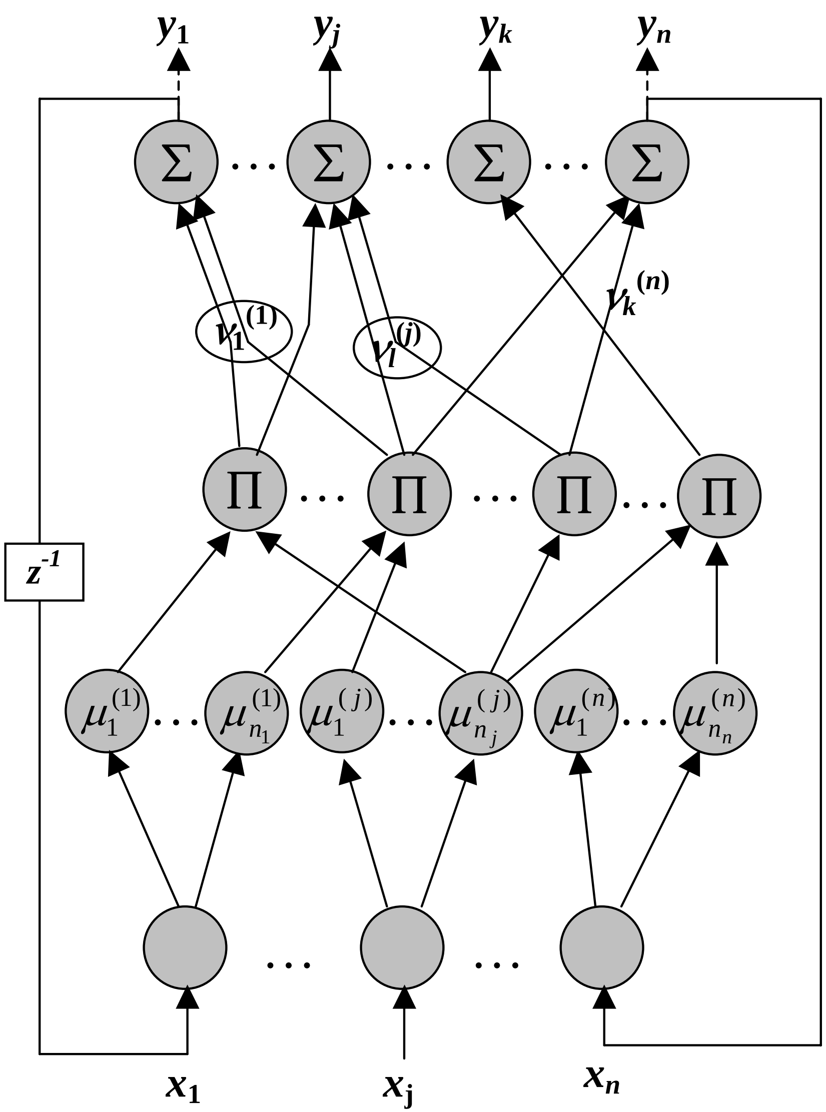



So, we are able to define the following functions:

$$s(t) = f(s(t_{i-1}), x(t_{i-1})), \quad y(t) = f(y(t_{i-1}), s(t_{i-1}))$$

These functions may be used to compute a system of two first order differential equations.

# A Hierarchical Hybrid Model




- based on recurrent hierarchical fuzzy system
- heuristics and GA applied for rule base learning
- optimization using gradient descent considering time delayed feed back
- interpretability enforced by:
  - coupled fuzzy sets
  - constraints

# A Hierarchical Hybrid Model (optimization)

Learning method minimizes error:

$$E = \sum_{t=0}^{T} E(t) = \sum_{t=0}^{T} \frac{1}{2} \sum_{k} (E_k(t))^2 \text{ with } E_k(t) = \begin{cases} o_k(t) - y_k(t) & \text{if the output } o_k \text{ for node } k \text{ at time } t \text{ is given,} \\ 0 & \text{otherwise.} \end{cases}$$

Gradient for parameter $p$:

$$\frac{\partial E(t)}{\partial p} = -\sum_{k} E_k(t) \frac{\partial y_k(t)}{\partial p} \quad \text{with} \quad \frac{\partial y_k(t)}{\partial p} = \frac{\partial y_k(t)}{\partial f_p(p)} \frac{\partial f(p)}{\partial p} + \sum_{r:\nu_{l_r}^{(k)} \in Con(r)} \frac{\partial y_k(t)}{\partial a_r(t)} \frac{\partial a_r(t)}{\partial p}$$

$$= \frac{\partial y_k(t)}{\partial f_p(p)} + \sum_{r:\nu_{l_r}^{(k)} \in Con(r)} \frac{\partial y_k(t)}{\partial a_r(t)} \frac{\partial a_r(t)}{\partial p}$$

$$\frac{\partial a_r(t)}{\partial p} = \frac{\partial a_r(t)}{\partial f_p(p)} \frac{\partial f(p)}{\partial p} + \sum_{i:\mu_{j_r}^{(i)} \in Ant(r)} \frac{\partial a_r(t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial p}$$

$$= \frac{\partial a_r(t)}{\partial f_p(p)} + \sum_{i:\mu_{j_r}^{(i)} \in Ant(r)} \frac{\partial a_r(t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial p}$$

# A Hierarchical Hybrid Model (optimization)

Finally we obtain:

$$\frac{\partial y_k(t)}{\partial p} = \frac{\partial y_k(t)}{\partial f_p(p)} + \sum_{r:\nu_{l_r}^{(k)} \in Con(r)} \left[ \frac{\partial y_k(t)}{\partial a_r(t)} \cdot \left( \frac{\partial a_r(t)}{\partial f_p(p)} + \sum_{i:\mu_{j_r}^{(i)} \in Ant(r)} \frac{\partial a_r(t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial p} \right) \right]$$

where

$$\frac{\partial x_i(t)}{\partial p} = \begin{cases} \dfrac{\partial y_j(t)}{\partial p} & \text{if } \dfrac{\partial x_i(t)}{\partial p} \text{ a hierarchical feedback } y_j(t), \\ \dfrac{\partial y_j(t-1)}{\partial p} & \text{if } \dfrac{\partial x_i(t)}{\partial p} \text{ a time delayed feedback } y_j(t-1), \\ \dfrac{\partial x_j(t-1)}{\partial p} & \text{if } \dfrac{\partial x_i(t)}{\partial p} \text{ a local feedback,} \\ 0 & \text{if } \dfrac{\partial x_i(t)}{\partial p} \text{ an external input.} \end{cases}$$

# A Hierarchical Hybrid Model (optimization)

## Updating parameter:

Online learning: $p(t+1) = p(t) + (1-\beta)\cdot\Delta p(t) + \beta\cdot\Delta p(t-1)$ , with

$$\Delta p(t) = -\eta_p \frac{\partial E(t)}{\partial p} = \eta_p \sum_k E_k(t) \frac{\partial y_k(t)}{\partial p}$$

Batch learning: $p^{(n+1)}(0) = p^{(n)}(0) + (1-\beta)\cdot\sum_{t=1}^{T}\Delta p^{(n)}(t) + \beta\cdot\sum_{t=1}^{T}\Delta p^{(n-1)}(t)$

$$\Delta p^{(n)}(t) = -\eta_p \frac{\partial E(t)}{\partial p} = \eta_p \sum_k E_k(t) \frac{\partial y_k(t)}{\partial p}$$

Time complexity: O( (#variables)$^4$ · (#fuzzy sets)$^3$ · (#rules) )

Memory complexity: O( (#variables)$^2$ · (#fuzzy sets) )

# Rule Base Learning

Learning a hierarchically structured rule base of local subsystems

- Here: Domains are partitioned
- Use of rule templates to define subsystems, e.g.:

  IF (x[t-1] LIKE ‘*S1') AND (v[t-1] LIKE '*S1') THEN (x[t] LIKE '*S0')
  IF (y[t] LIKE '*') AND (v[t-1] LIKE '*S2') THEN (v[t] LIKE '*S0').

- Two learning methods:
  - Heuristics: Iterative creation of rules
  - Genetic Algorithm

# Rule Base Learning

## Heuristics:

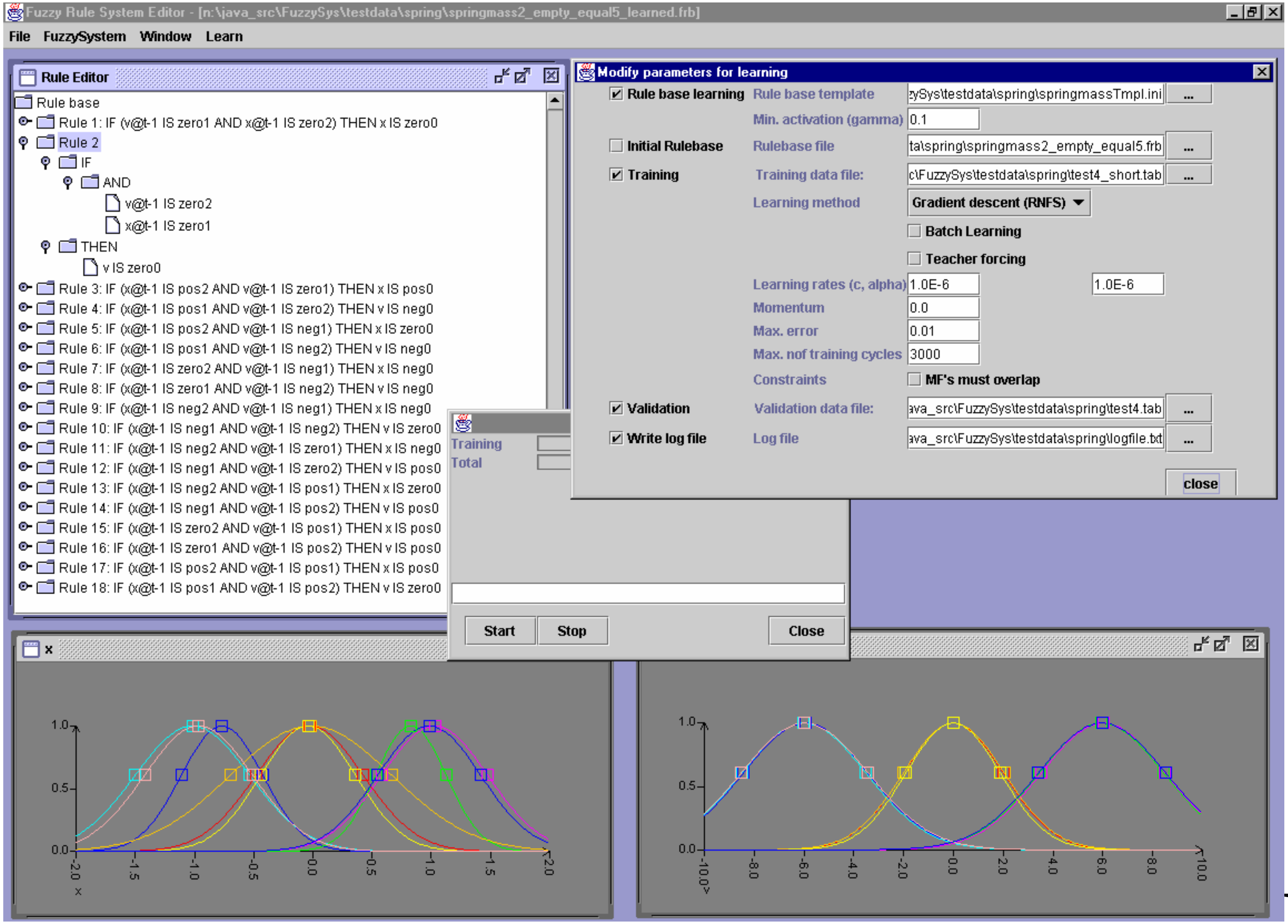
- Two learning parts:
  - Rule base learning (iterative creation of rules)
  - Re-assigning consequents

## Genetic Algorithm:

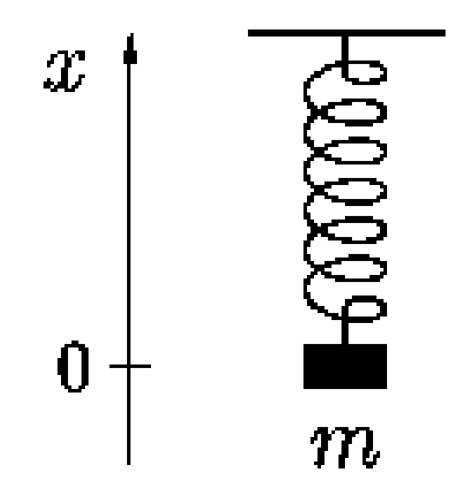
- Assume full rule base
- Each Chromosome encodes complete rule base (Pittsburgh approach)

| | Ordered Rule Base | Chromosome |
|---|---|---|
| Sub system 1 | IF (x[t-1] IS neg2 AND v[t-1] IS neg1) THEN x IS neg0 | 1 |
| | IF (x[t-1] IS neg2 AND v[t-1] IS zero1) THEN x IS neg0 | 1 |
| | IF (x[t-1] IS neg2 AND v[t-1] IS pos1) THEN x IS zero0 | 2 |
| | IF (x[t-1] IS zero2 AND v[t-1] IS neg1) THEN x IS neg0 | 1 |
| | IF (x[t-1] IS zero2 AND v[t-1] IS zero1) THEN x IS zero0 | 2 |
| | IF (x[t-1] IS zero2 AND v[t-1] IS pos1) THEN x IS pos0 | 3 |
| | IF (x[t-1] IS pos2 AND v[t-1] IS neg1) THEN x IS zero0 | 1 |
| | IF (x[t-1] IS pos2 AND v[t-1] IS zero1) THEN x IS pos0 | 3 |
| | IF (x[t-1] IS pos2 AND v[t-1] IS pos1) THEN x IS pos0 | 3 |
| Sub system 2 | IF (x[t-1] IS neg1 AND v[t-1] IS neg2) THEN v IS zero0 | 2 |
| | IF (x[t-1] IS neg1 AND v[t-1] IS zero2) THEN v IS pos0 | 3 |
| | IF (x[t-1] IS neg1 AND v[t-1] IS pos2) THEN v IS pos0 | 3 |
| | IF (x[t-1] IS zero1 AND v[t-1] IS neg2) THEN v IS neg0 | 1 |
| | IF (x[t-1] IS zero1 AND v[t-1] IS zero2) THEN v IS zero0 | 2 |
| | IF (x[t-1] IS zero1 AND v[t-1] IS pos2) THEN v IS pos0 | 3 |
| | IF (x[t-1] IS pos1 AND v[t-1] IS neg2) THEN v IS neg0 | 1 |
| | IF (x[t-1] IS pos1 AND v[t-1] IS zero2) THEN v IS neg0 | 1 |
| | IF (x[t-1] IS pos1 AND v[t-1] IS pos2) THEN v IS zero0 | 2 |

# Software Implementation



# A Simple Physical System: Mass On a Spring




With the physical laws

$$F = -c \cdot \Delta l = -c \cdot x \quad \text{(Hooke's law)}$$

$$F = m \cdot a = m \cdot \ddot{x} \quad \text{(Newton's second law)}$$

we can describe the system by a second order differential equation

$$\ddot{x} = -\frac{c}{m} x \text{, with } \; x(0) = x_0, v(0) = \dot{x}(0) = 0$$

or by a system of two first order differential equations

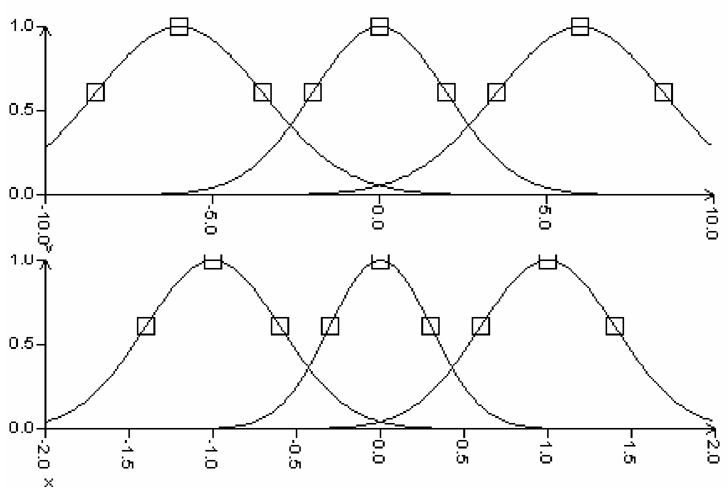
$$\dot{x} = v \quad \text{and} \quad \dot{v} = -\frac{c}{m} x.$$

# Application Example (Spring-mass model)

Training data: Simulation of the modell for a period of 20 sec.

Initial partitioning:

(FS's: *neg0, neg1, neg2,*
*zero0, zero1, zero2,*
*pos0, pos1, pos2)*



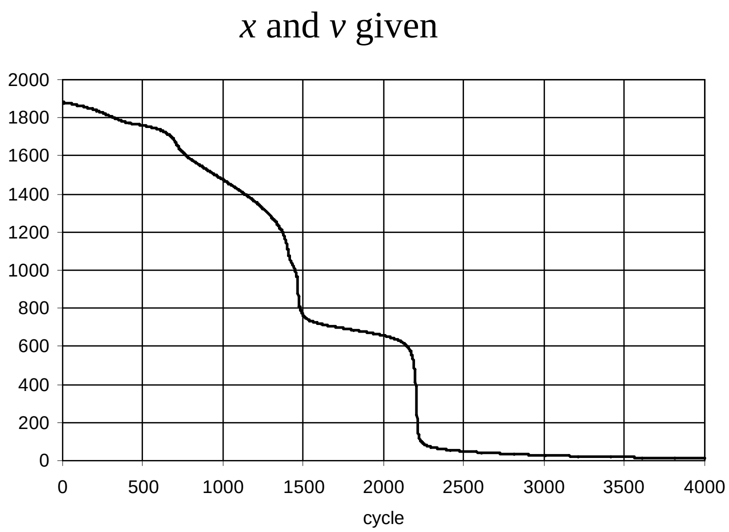

Templates used for rule base learning:

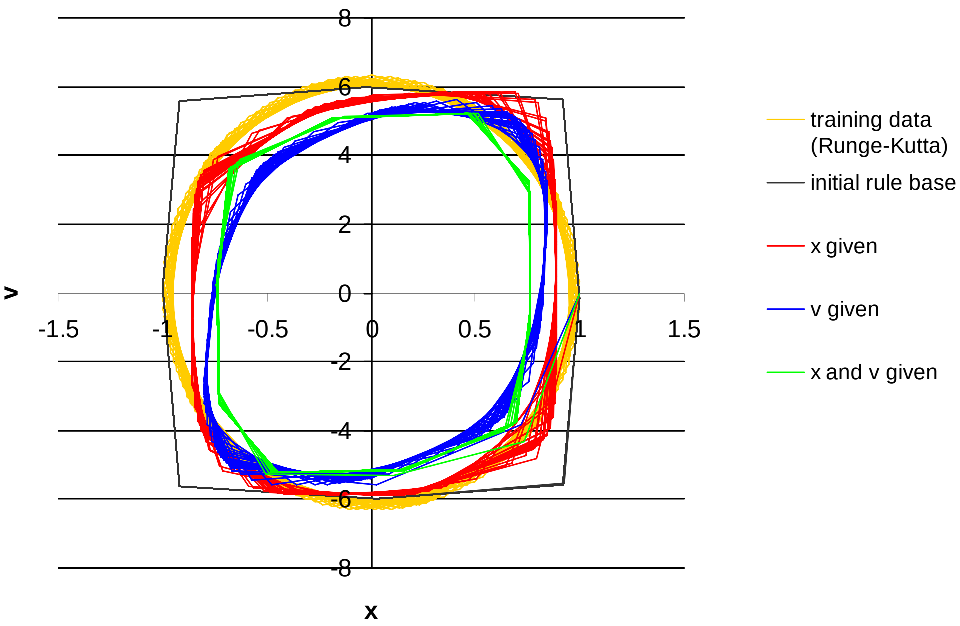
IF (x[t-1] LIKE '*2') AND (v[t-1] LIKE '*1') THEN (x[t-0] LIKE '*0')
IF (x[t-1] LIKE '*1') AND (v[t-1] LIKE '*2') THEN (v[t-0] LIKE '*0')

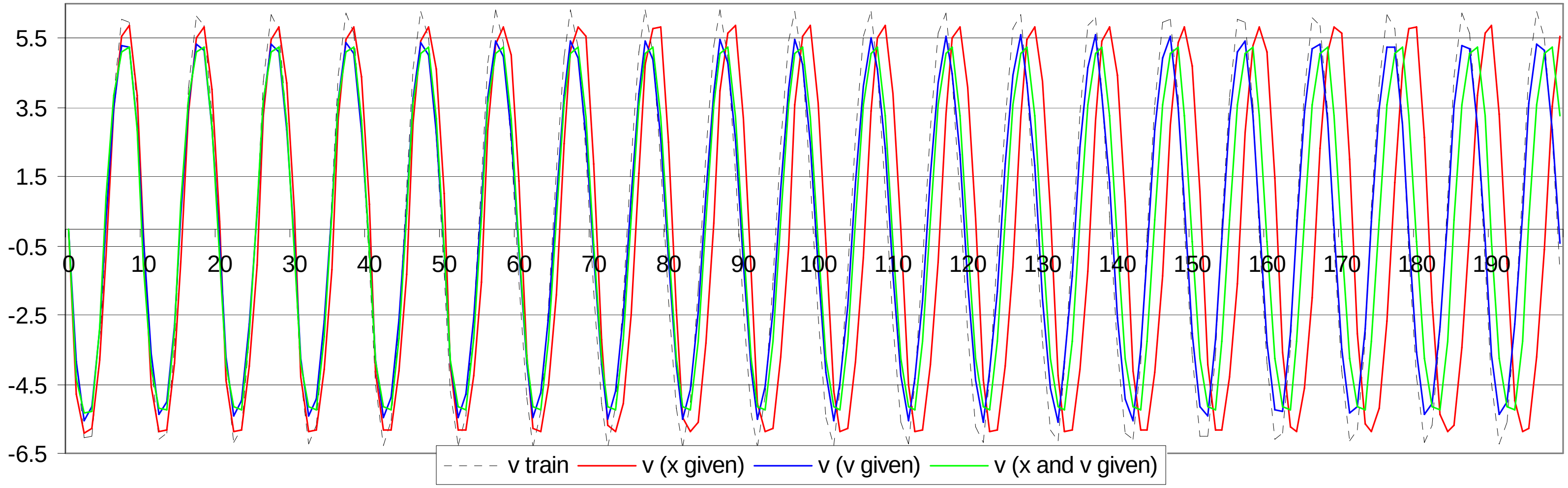
Initial rules (not covered by simulation):

Rule 1: IF (x@t-1 IS zero2 AND v@t-1 IS zero1) THEN x IS zero0
Rule 2: IF (x@t-1 IS zero1 AND v@t-1 IS zero2) THEN v IS zero0

# Application Example (Spring-mass model)



# Application Example (Spring-mass model)

| | | | |
|---|---|---|---|
| Rule 9: | IF (x[t-1] IS neg2 | AND v[t-1] IS neg1) | THEN x IS neg0 |
| Rule 11: | IF (x[t-1] IS neg2 | AND v[t-1] IS zero1) | THEN x IS neg0 |
| Rule 13: | IF (x[t-1] IS neg2 | AND v[t-1] IS pos1) | THEN x IS zero0 |
| Rule 7: | IF (x[t-1] IS zero2 | AND v[t-1] IS neg1) | THEN x IS neg0 |
| Rule 1: | IF (x[t-1] IS zero2 | AND v[t-1] IS zero1) | THEN x IS zero0 |
| Rule 15: | IF (x[t-1] IS zero2 | AND v[t-1] IS pos1) | THEN x IS pos0 |
| Rule 5: | IF (x[t-1] IS pos2 | AND v[t-1] IS neg1) | THEN x IS zero0 |
| Rule 3: | IF (x[t-1] IS pos2 | AND v[t-1] IS zero1) | THEN x IS pos0 |
| Rule 17: | IF (x[t-1] IS pos2 | AND v[t-1] IS pos1) | THEN x IS pos0 |

| | | | |
|---|---|---|---|
| Rule 10: | IF (x[t-1] IS neg1 | AND v[t-1] IS neg2) | THEN v IS zero0 |
| Rule 12: | IF (x[t-1] IS neg1 | AND v[t-1] IS zero2) | THEN v IS pos0 |
| Rule 14: | IF (x[t-1] IS neg1 | AND v[t-1] IS pos2) | THEN v IS pos0 |
| Rule 8: | IF (x[t-1] IS zero1 | AND v[t-1] IS neg2) | THEN v IS neg0 |
| Rule 2: | IF (x[t-1] IS zero1 | AND v[t-1] IS zero2) | THEN v IS zero0 |
| Rule 16: | IF (x[t-1] IS zero1 | AND v[t-1] IS pos2) | THEN v IS pos0 |
| Rule 6: | IF (x[t-1] IS pos1 | AND v[t-1] IS neg2) | THEN v IS neg0 |
| Rule 4: | IF (x[t-1] IS pos1 | AND v[t-1] IS zero2) | THEN v IS neg0 |
| Rule 18: | IF (x[t-1] IS pos1 | AND v[t-1] IS pos2) | THEN v IS zero0 |

$$x(t_i) = x(t_{i-1}) + \Delta t \cdot v(t_{i-1}) \qquad v(t_i) = v(t_{i-1}) - \frac{c}{m} \Delta t \cdot x(t_{i-1})$$

# Recurrent Neuro-Fuzzy Systems (Cooperative approach)

**Motivation/Problem:** Modeling, identification and simulation of viscoelastic models in virtual reality applications

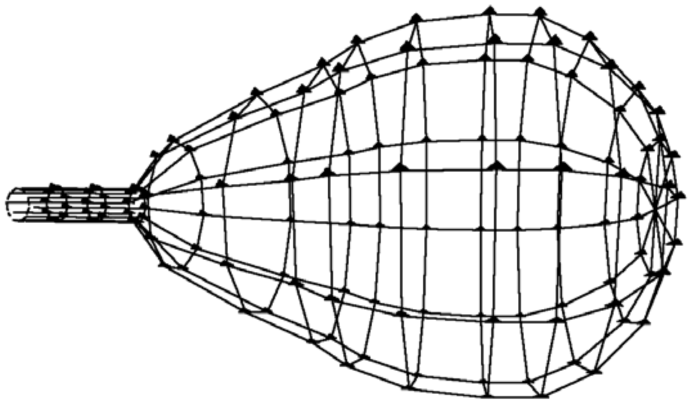
**Existing approaches:** Model creation usually expensive, cannot be optimized by measured data

**Idea:** Support developer with a generic framework for model creation and definition / optimization of its parameters:

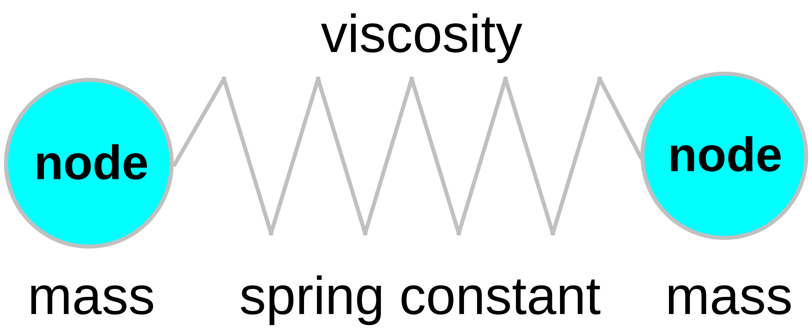
- Enable use of standard 3D models (triangular meshes)
- Fuzzy system for interactive definition of model parameters and initialization of the learning process
- Recurrent neural network architecture for simulation and model optimization if measured data is available

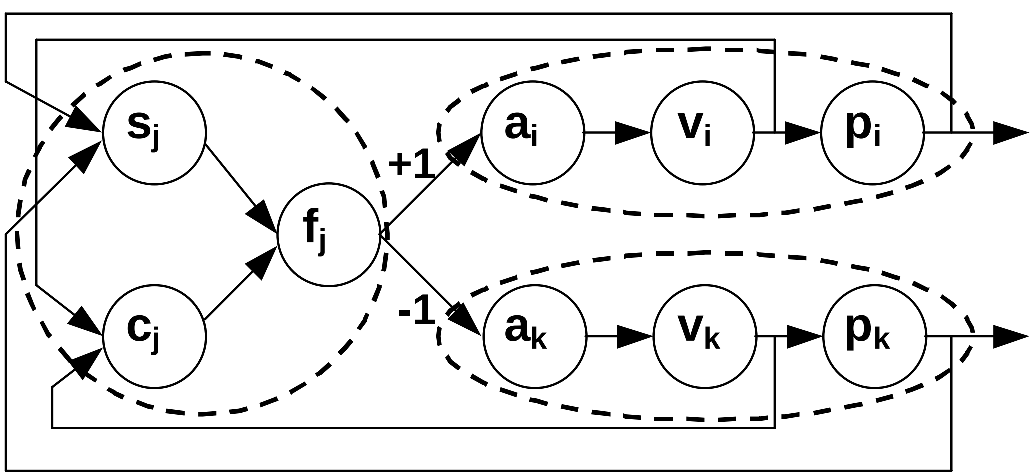
➔ Cooperative neuro-fuzzy architecture

# RCNN describing viscoelastic models



$$F = c \cdot \Delta l - d \cdot l \qquad F = m \cdot a = m \cdot \ddot{x}$$




$$\vec{o}_p(t) = \vec{f}_p(\vec{net}_p(t)) = t_c \vec{o}_v(t) + \vec{o}_p(t-1)$$

$$\vec{o}_v(t) = \vec{f}_v(\vec{net}_v(t)) = t_c \vec{o}_a(t) + \vec{o}_v(t-1)$$

$$\vec{o}_a(t) = \vec{f}_a(\vec{net}_a(t)) = w_m\left(\sum_i w_i \vec{o}_{f_1}(t) + ext_f(t)\right)$$

$$\vec{o}_f(t) = \vec{f}_f(\vec{net}_f(t)) = c_c \vec{o}_c(t) + s_c \vec{o}_s(t)$$

$$\vec{o}_s(t) = \vec{f}_s(\vec{net}_s(t)) = \vec{f}_s(\vec{o}_{p_1}(t) - \vec{o}_{p_2}(t))$$

$$\vec{o}_c(t) = \vec{f}_c(\vec{net}_c(t)) = \vec{f}_c(\vec{o}_{v_1}(t) - \vec{o}_{v_2}(t))$$

# Parameter determination (Learning considerations)

- Requirements:
  - **Learning by use of time series data (node positions)**
  - **Changing external forces**
  - **Missing data for (inner) nodes**
  - **Missing data for „intermediate“ time steps**
- Problems:
  - **Large number of time steps between attractors**
  - **Great time delay between weight modification and effect on positions**
  - **Missing data**

➔**Combination of BPTT and RTRL with teacher forcing**

# Parameter determination (Initialization)

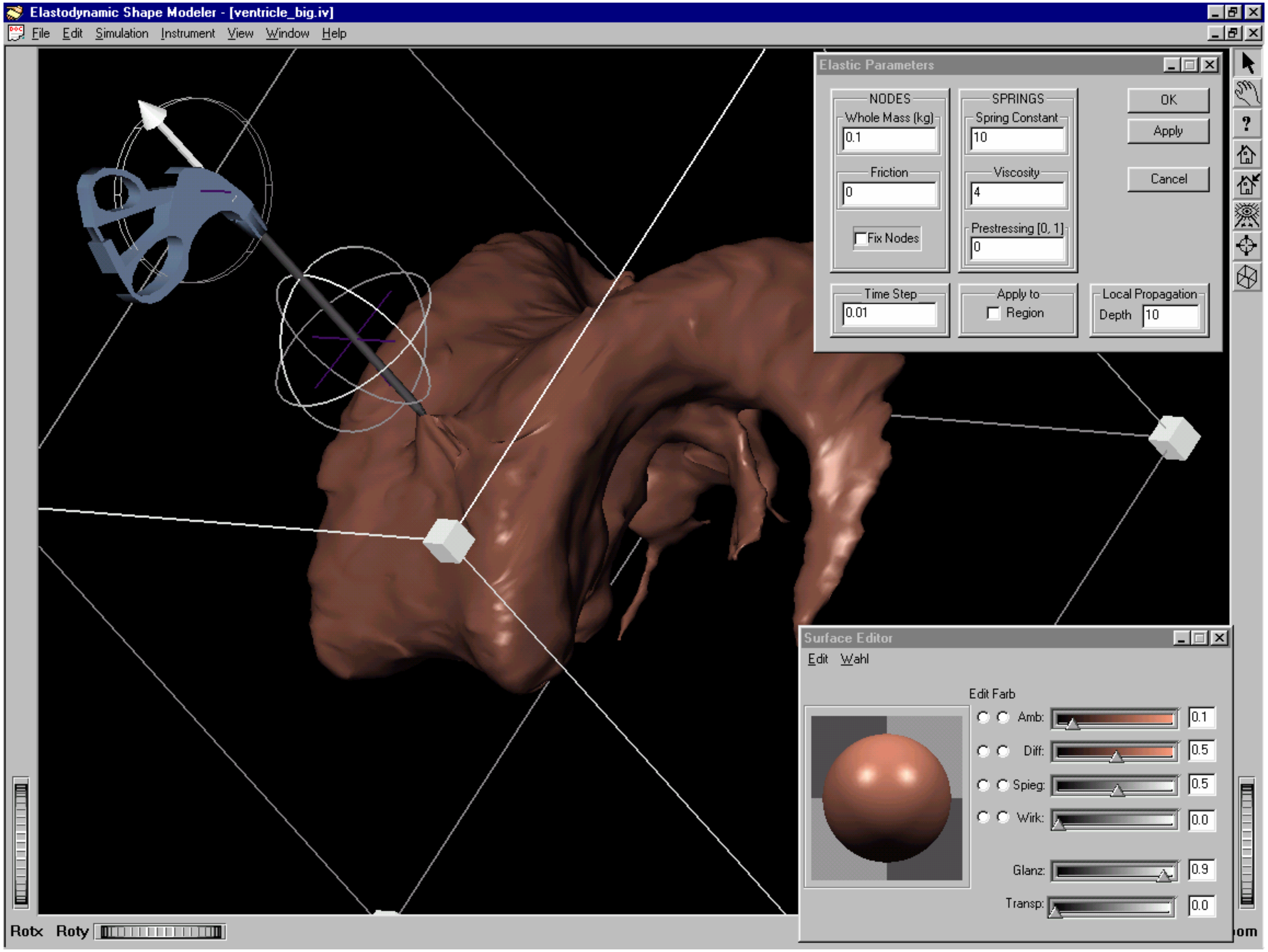
- **Initialization by „real“ physical parameters**
- **Initialization by use of a fuzzy system:**

  **Fuzzy system describes the relations between vague knowledge of the object and the network parameters**

*if* hardness *is* soft *and* elasticity *is* big
*then* spring constant *is* low
*if* shiftability *is* high
*then* spring constant *is* very low *and* viscosity *is* high
*if* consistency *is* chapped
*then* fraction force *is* low *and* spring constant *is* very high
*if* consistency *is* slightly hard
*then* spring constant *is* slightly high *and* viscosity *is* low

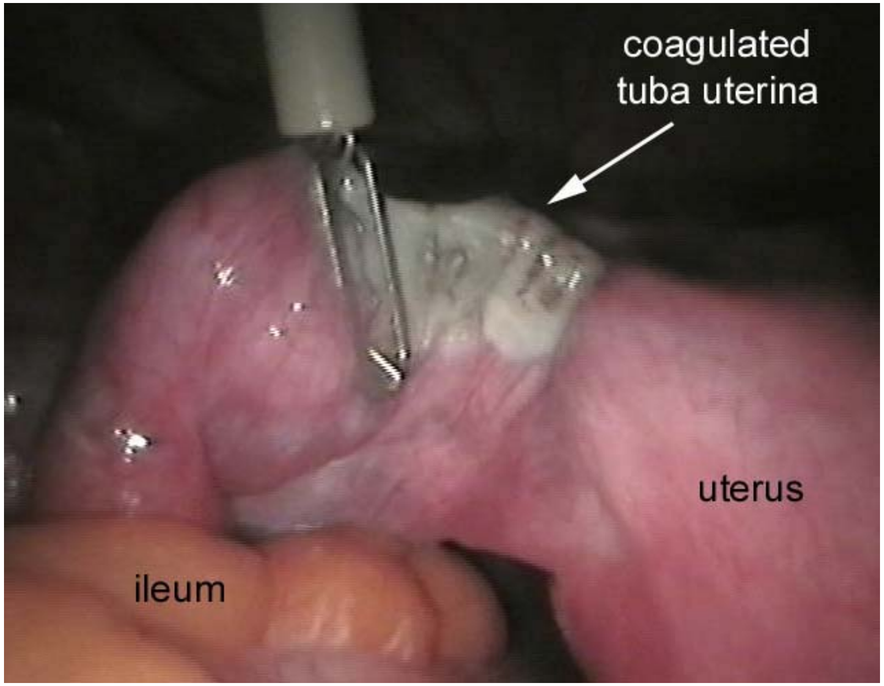
**Some fuzzy rules for the description of tissue**

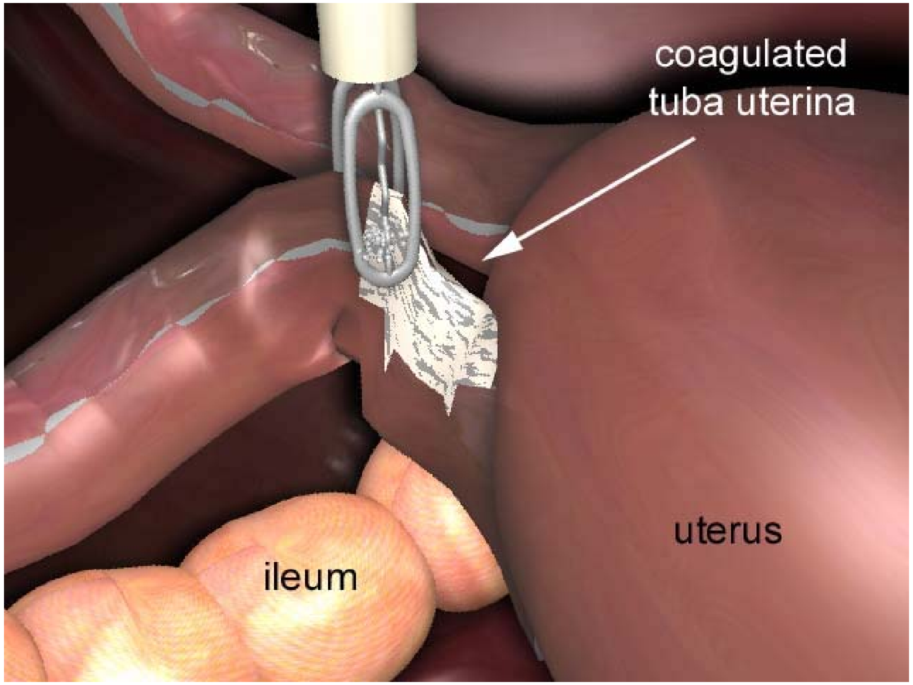
# Parameter determination (Initialization)

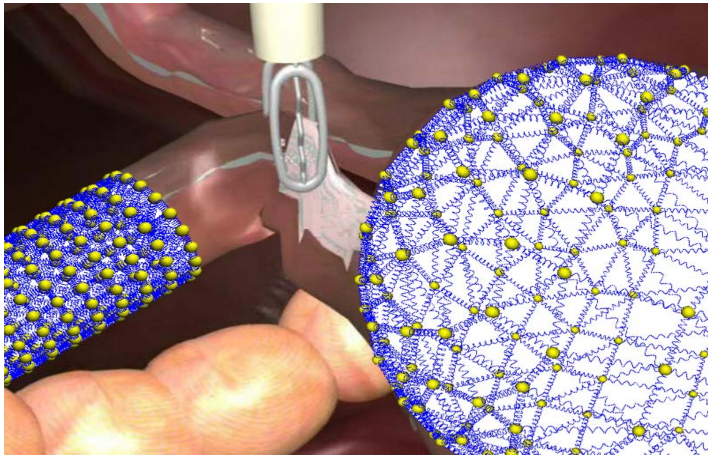


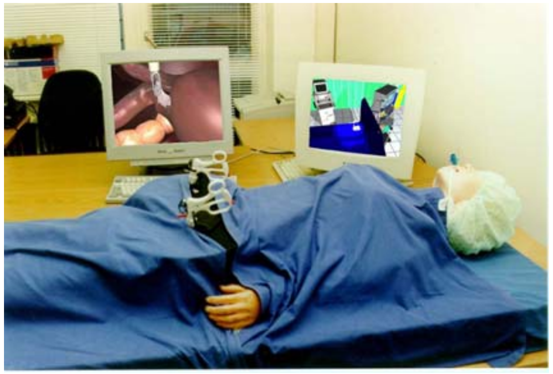

Screenshot of the tool 'Elastodynamic Shape Modeler'

# SUSILAP-G (SUrgical SImulator for LAParoscopy in Gynaecology)





# Conclusions

- Recurrent fuzzy systems can be used to approximate dynamic systems
- Proposed recurrent neuro-fuzzy system can be used to:
  - optimize recurrent and/or hierarchical fuzzy systems
  - learn rule base by use of rule templates
- Constraints must be used more carefully than in feed-forward systems