technische universität
dortmund

Computational Intelligence

Winter Term 2020/21

Prof. Dr. Günter Rudolph

Lehrstuhl für Algorithm Engineering (LS 11)

Fakultät für Informatik

TU Dortmund

Plan for Today

- Recurrent Neural Networks
  - Excursion: Nonlinear Dynamics
  - Recurrent Models
  - Training

S state space with states $s \in S$

Θ parameter space with parameters $\theta \in \Theta$

$s^{(t)}$ is a state $\in S$ at time $t \in \mathbb{N}_0$

$f : S \times \Theta \to S$ transition function

$\to$ dynamical system $s^{(t+1)} = f(s^{(t)}, \theta)$ (*) **recurrence relation**

$$s^{(t)} = f^t(s^{(0)}, \theta) = \underbrace{f \circ \cdots \circ f}_{t \text{ times}}(s^{(0)}, \theta) = \underbrace{f_\theta(f_\theta(f_\theta(\cdots f_\theta}_{t \text{ times}}(s^{(0)})))); \quad f_\theta(s) = f(s, \theta)$$

D: s^* is called **stationary point** / **fixed point** / **steady state of (*) if** $s^* = f(s^*)$

D: stationary point s^* is **locally asymptotical stable (l.a.s.)** if

$$\exists \varepsilon > 0 : \forall s^{(0)} \in B_\varepsilon(s^*) : \lim_{t \to \infty} s^{(t)} = s^*$$

T: Let f be differentiable. Then s is l.a.s. if $|f'(s)| < 1$, and unstable if $|f'(s)| > 1$.

Remark: D: $s \in S$ is **recurrent** if $\forall \varepsilon > 0 : \exists t > 0 : f^t(s) \in B_\varepsilon(s)$ infinitly often (i.o.)

examples

- <u>linear case:</u> $f(x) = a\,x + b \qquad a, b \in \mathbb{R}$

  fixed points: $x = f(x) = a\,x + b \quad \Rightarrow \quad x = \frac{b}{1-a}$ if $a \neq 1$

  stability: $f'(x) = a \qquad \Rightarrow |f'(x^*)| = |a| < 1$ l.a.s., $|a| > 1$ unstable

- <u>nonlinear case:</u> $f(x) = r\,x\,(1-x) \quad r \in (0, 4] \quad x \in (0, 1)$ logistic map

  fixed points: $x = f(x) = r\,x\,(1-x) \quad \Rightarrow \quad x = 0$ or $x = 1 - \frac{1}{r} = \frac{r-1}{r}$

  stability: $f'(x) = r - 2r\,x$

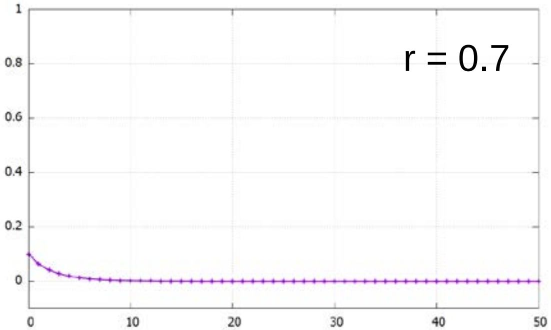
  $|f'(0)| = r < 1 \quad \Rightarrow$ l.a.s. also for $r = 1$ since $x < f(x)$ for $x < \frac{1}{2}$

  $|f'(\frac{r-1}{r})| = |2 - r| < 1 \Leftrightarrow 1 < r < 3$ l.a.s.

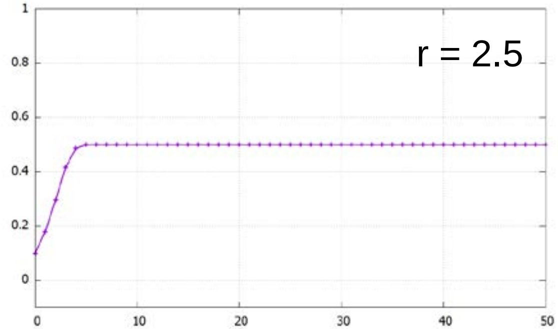
  $r \in [3, 1 + \sqrt{6})$ oscillation between 2 values
  $r \in [1 + \sqrt{6}, 3.54\ldots)$ oscillation between 4 values
  $\vdots$ 8, 16, 32, ...
  $r > 3.56995\ldots$ deterministic chaos

→ predicting a nonlinear dynamic system may be impossible!

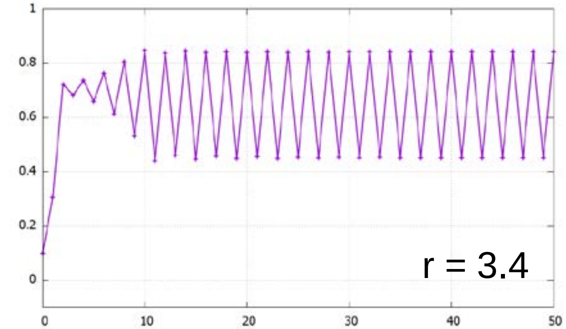
logistic map starting at x = 0.1



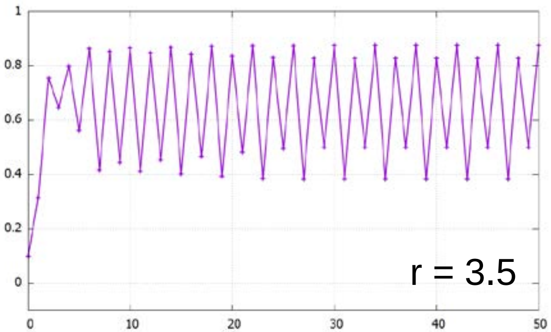
stable fixed point at x = 0



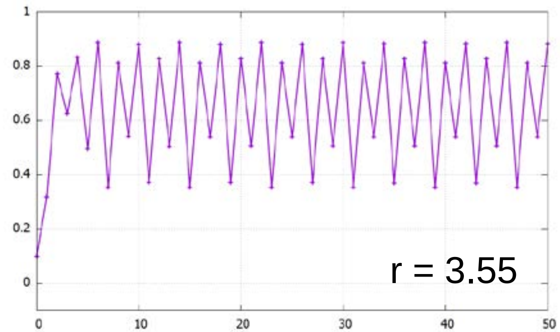
stable fixed point at x = 0.5



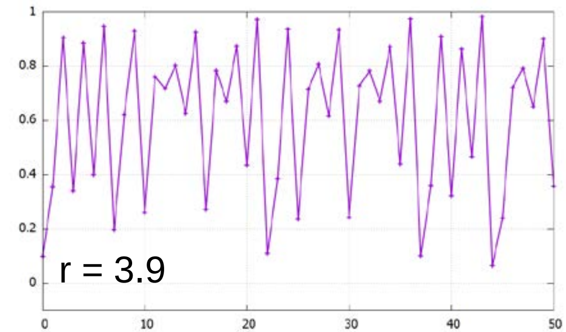
peridic orbit of size 2



periodic orbit of size 4



periodic orbit of size 8



deterministic chaos

extensions

- dynamical system with inputs

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta)$$

input at time $t \in \mathbb{N}$

- dynamical system with inputs and outputs

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta_f)$$

$$o^{(t)} = g(s^{(t)}; \theta_g)$$

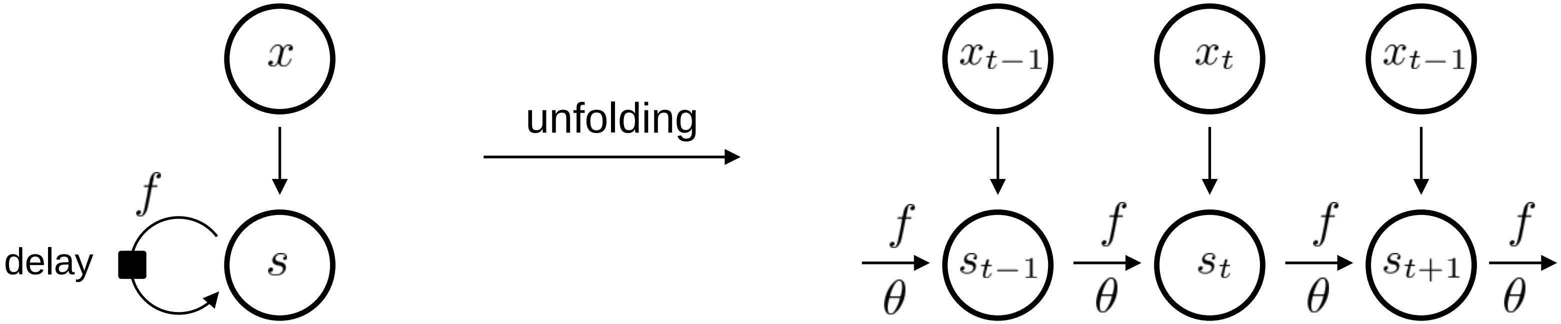
output at time $t \in \mathbb{N}$

describes a **recurrent** neural network (RNN)

unfolding

- finite input sequence
  $\Rightarrow$ can unfold RNN completely to (deep) feed forward network

- infinite input sequence
  $\Rightarrow$ can unfold RNN only finitely many steps into the past
  $\Rightarrow$ assumption: behavior mainly depends on few inputs in the past
  (i.e., **no** long-term dependencies)



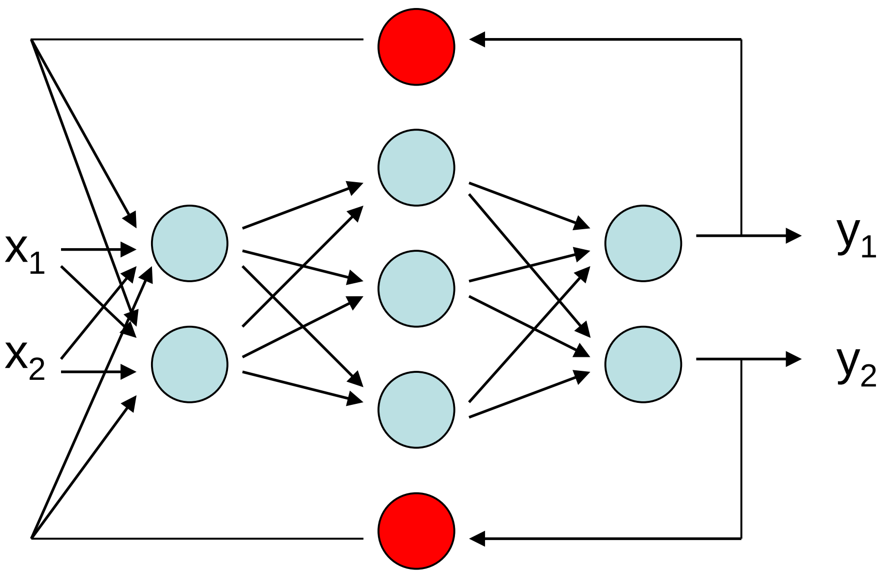
remark: parameters θ in unfolded network are shared
otherwise with θ_t overfitting becomes very likely!

- Jordan network (1983)

$$
\begin{aligned}
s_t &= f(s_{t-1}, x_t; W, U, b) \\
&= \sigma(W x_t + U \hat{y}_{t-1} + b)
\end{aligned}
$$

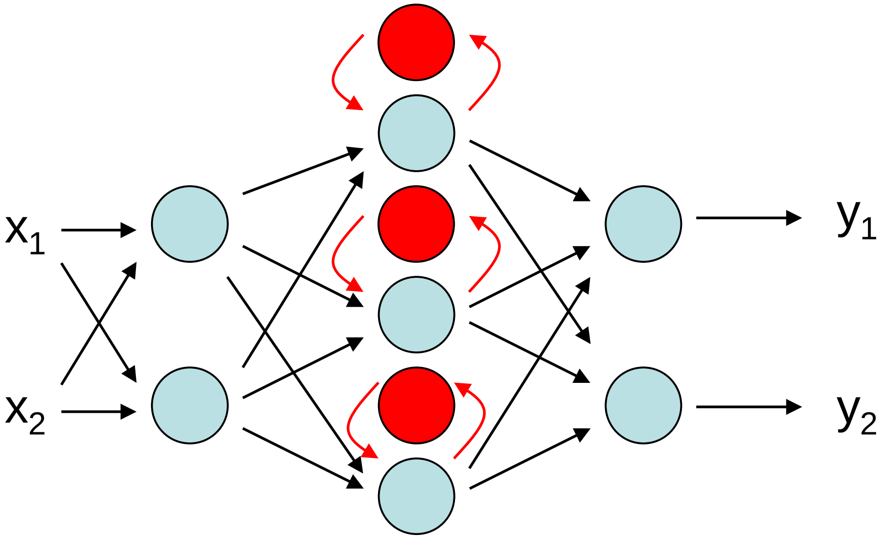
$$
\begin{aligned}
o_t &= g(s_t; V, c) \\
&= V s_t + c
\end{aligned}
$$

$$
\hat{y}_t = a(o_t)
$$

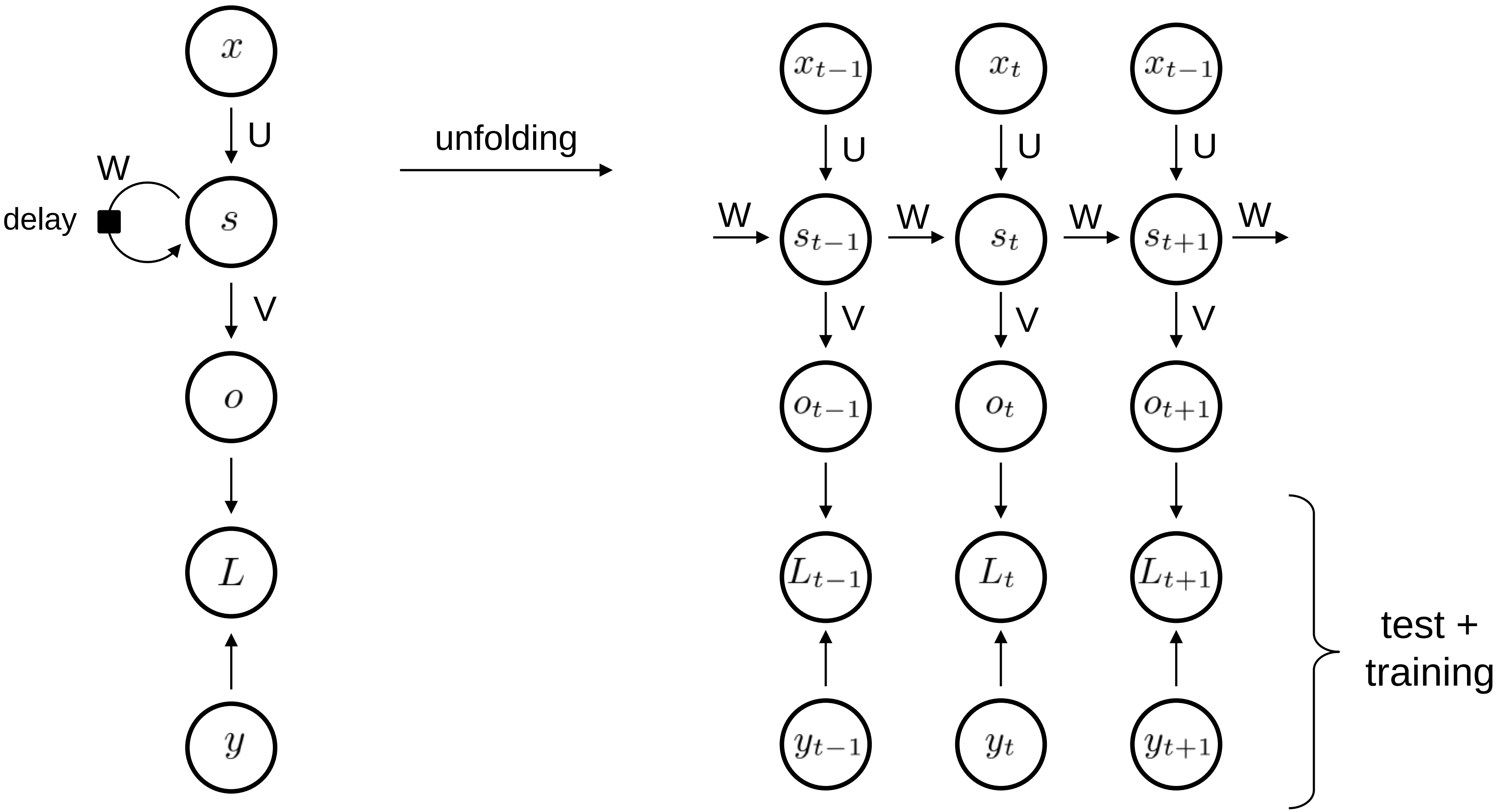


- Elman network (1990)

$$
\begin{aligned}
s_t &= \sigma(W x_t + U s_{t-1} + b) \\
o_t &= V s_t + c \\
\hat{y}_t &= a(o_t)
\end{aligned}
$$



test / training mode



loss per input $L(\hat{y}, y) = \|\hat{y} - y\|_2^2$ where $\hat{y} = \text{SOFTMAX}(o)$

training? → **backpropagation through time (BPTT)**

- works on unfolded network for a finite input sequence $x^{(1)}, \ldots, x^{(\tau)}$
- some adaption to BP necessary, since many parameters are shared

reduces #params and overfitting

- “straightforward“ (but tedious + error-prone if done manually)

  → use method from your software library!
- in principle: gradient descent on loss function

LSTM network (1997f.)

LSTM = long short-term memory

<u>so far:</u> no long-term dependencies

<u>now:</u> “remember the important stuff and forget the rest“ [Cha18, p.89]

concept: two versions of the past

1. selective long-term memory
2. short term memory

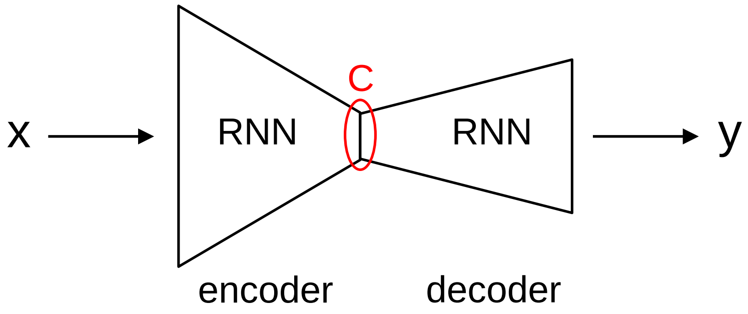
historic/standard RNN forget too quickly

- has the ability to learn long-term dependencies
- technical problem: vanishing gradient

encoder / decoder architecture (~2014)

so far: length of input = length of output

now: different lengths → typical situation e.g. in language translation



context C = semantic summary of input sequence

- **encoder**: RNN reading input sequence of length τ_x

  delivers ‘context‘ C as function of final layer

- **decoder**: RNN reading context C

  delivers delivers output sequence of length τ_y as function of final layer