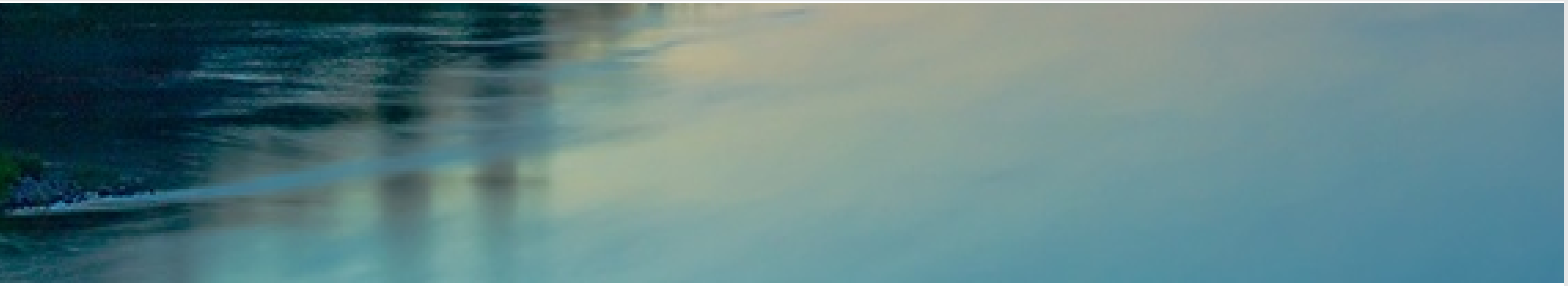


Neural Networks

Prof. Dr. Rudolf Kruse

Computational Intelligence Group
Faculty for Computer Science
kruse@iws.cs.uni-magdeburg.de

Recurrent Neural Networks

Recurrent Networks: Cooling Law

A body of temperature ϑ_0 that is placed into an environment with temperature ϑ_A.

The cooling/heating of the body can be described by **Newton's cooling law**:

$$\frac{\mathrm{d}\vartheta}{\mathrm{d}t} = \dot{\vartheta} = -k(\vartheta - \vartheta_A).$$

Exact analytical solution:

$$\vartheta(t) = \vartheta_A + (\vartheta_0 - \vartheta_A)e^{-k(t-t_0)}$$

Approximate solution with **Euler-Cauchy polygon courses**:

$$\vartheta_1 = \vartheta(t_1) = \vartheta(t_0) + \dot{\vartheta}(t_0)\Delta t = \vartheta_0 - k(\vartheta_0 - \vartheta_A)\Delta t.$$

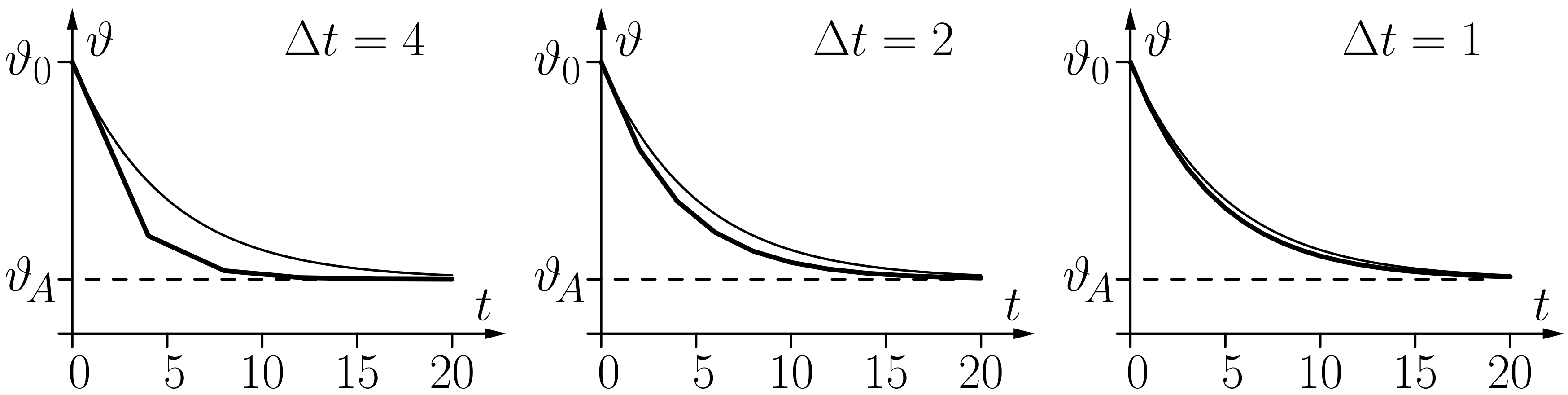
$$\vartheta_2 = \vartheta(t_2) = \vartheta(t_1) + \dot{\vartheta}(t_1)\Delta t = \vartheta_1 - k(\vartheta_1 - \vartheta_A)\Delta t.$$

General recursive formula:

$$\vartheta_i = \vartheta(t_i) = \vartheta(t_{i-1}) + \dot{\vartheta}(t_{i-1})\Delta t = \vartheta_{i-1} - k(\vartheta_{i-1} - \vartheta_A)\Delta t$$

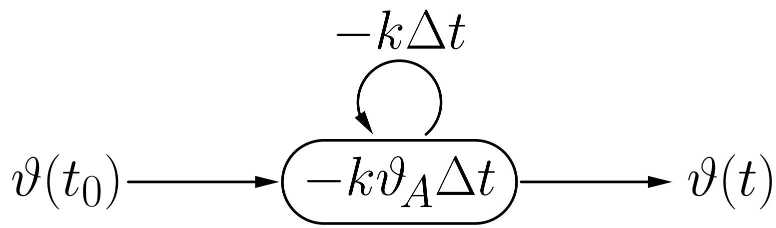
Recurrent Networks: Cooling Law

Euler–Cauchy polygon courses for different step widths:



The thin curve is the exact analytical solution.

Recurrent neural network:



Recurrent Networks: Cooling Law

More formal derivation of the recursive formula:

Replace differential quotient by **forward difference**

$$\frac{\mathrm{d}\vartheta(t)}{\mathrm{d}t} \approx \frac{\Delta\vartheta(t)}{\Delta t} = \frac{\vartheta(t+\Delta t)-\vartheta(t)}{\Delta t}$$

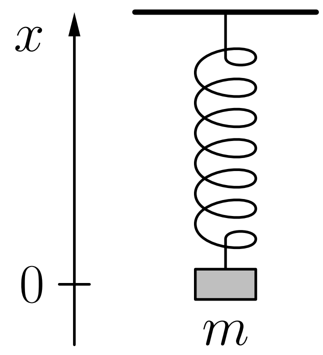
with sufficiently small Δt. Then it is

$$\vartheta(t+\Delta t)-\vartheta(t) = \Delta\vartheta(t) \approx -k(\vartheta(t)-\vartheta_A)\Delta t,$$

$$\vartheta(t+\Delta t)-\vartheta(t) = \Delta\vartheta(t) \approx -k\Delta t\vartheta(t)+k\vartheta_A\Delta t$$

and therefore

$$\vartheta_i \approx \vartheta_{i-1} - k\Delta t\vartheta_{i-1} + k\vartheta_A\Delta t.$$



Governing physical laws:

- **Hooke's law**: $F = c\Delta l = -cx$ (c is a spring dependent constant)
- **Newton's second law**: $F = ma = m\ddot{x}$ (force causes an acceleration)

Resulting differential equation:

$$m\ddot{x} = -cx \qquad \text{or} \qquad \ddot{x} = -\frac{c}{m}x.$$

General analytical solution of the differential equation:

$$x(t) = a \sin(\omega t) + b \cos(\omega t)$$

with the parameters

$$\omega = \sqrt{\frac{c}{m}}, \qquad \begin{aligned} a &= x(t_0) \sin(\omega t_0) \;+\; v(t_0) \cos(\omega t_0), \\ b &= x(t_0) \cos(\omega t_0) \;-\; v(t_0) \sin(\omega t_0). \end{aligned}$$

With given initial values $x(t_0) = x_0$ and $v(t_0) = 0$ and
the additional assumption $t_0 = 0$ we get the simple expression

$$x(t) = x_0 \cos\left(\sqrt{\frac{c}{m}}\, t\right).$$

Recurrent Networks: Mass on a Spring

Turn differential equation into two coupled equations:

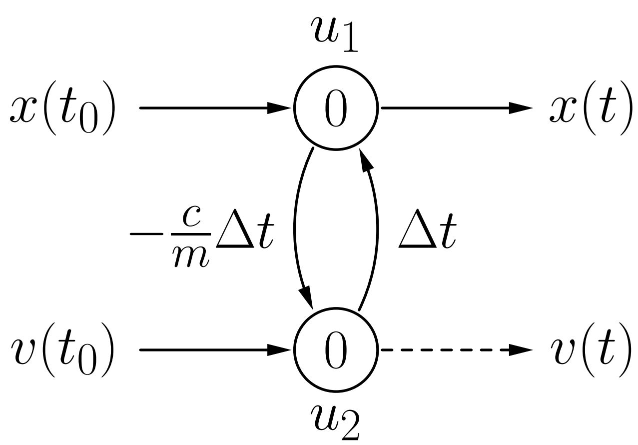
$$\dot{x} = v \qquad \text{and} \qquad \dot{v} = -\frac{c}{m}x.$$

Approximate differential quotient by forward difference:

$$\frac{\Delta x}{\Delta t} = \frac{x(t+\Delta t) - x(t)}{\Delta t} = v \qquad \text{and} \qquad \frac{\Delta v}{\Delta t} = \frac{v(t+\Delta t) - v(t)}{\Delta t} = -\frac{c}{m}x$$

Resulting recursive equations:

$$\begin{aligned} x(t_i) &= x(t_{i-1}) + \Delta x(t_{i-1}) &= x(t_{i-1}) + \Delta t \cdot v(t_{i-1}) \qquad \text{and} \\ v(t_i) &= v(t_{i-1}) + \Delta v(t_{i-1}) &= v(t_{i-1}) - \frac{c}{m}\Delta t \cdot x(t_{i-1}). \end{aligned}$$



Neuron u_1: $\quad f_{\text{net}}^{(u_1)}(v, w_{u_1u_2}) = w_{u_1u_2}v = -\dfrac{c}{m}\Delta t\ v \quad$ and

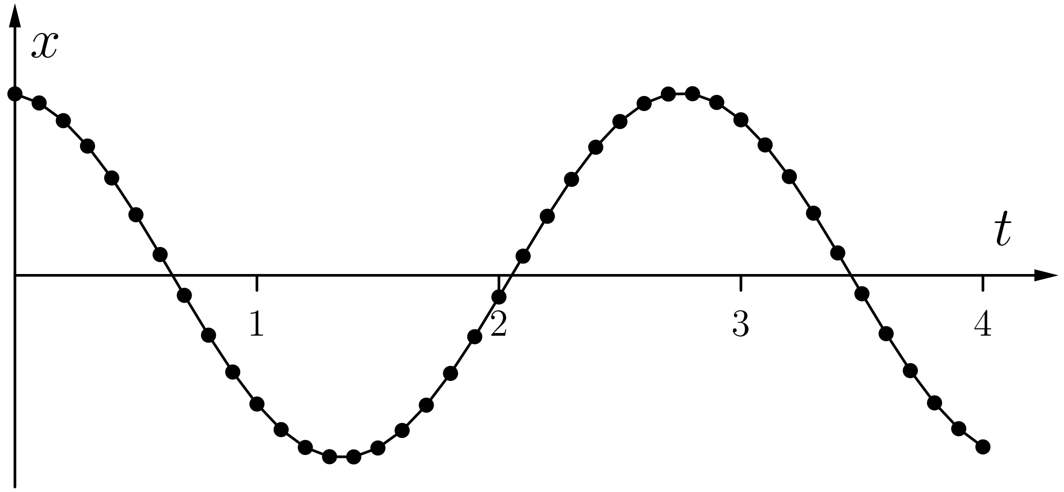
$$f_{\text{act}}^{(u_1)}(\text{act}_{u_1}, \text{net}_{u_1}, \theta_{u_1}) = \text{act}_{u_1} + \text{net}_{u_1} - \theta_{u_1},$$

Neuron u_2: $\quad f_{\text{net}}^{(u_2)}(x, w_{u_2u_1}) = w_{u_2u_1}x = \Delta t\ x \quad$ and

$$f_{\text{act}}^{(u_2)}(\text{act}_{u_2}, \text{net}_{u_2}, \theta_{u_2}) = \text{act}_{u_2} + \text{net}_{u_2} - \theta_{u_2}.$$

Some computation steps of the neural network:

t	v	x
0.0	0.0000	1.0000
0.1	−0.5000	0.9500
0.2	−0.9750	0.8525
0.3	−1.4012	0.7124
0.4	−1.7574	0.5366
0.5	−2.0258	0.3341
0.6	−2.1928	0.1148



- The resulting curve is close to the analytical solution.
- The approximation gets better with smaller step width.

General representation of explicit n-th order differential equation:

$$x^{(n)} = f(t, x, \dot{x}, \ddot{x}, \ldots, x^{(n-1)})$$

Introduce $n-1$ intermediary quantities

$$y_1 = \dot{x}, \qquad y_2 = \ddot{x}, \qquad \ldots \qquad y_{n-1} = x^{(n-1)}$$

to obtain the system

$$\begin{aligned} \dot{x} &= y_1, \\ \dot{y}_1 &= y_2, \\ &\vdots \\ \dot{y}_{n-2} &= y_{n-1}, \\ \dot{y}_{n-1} &= f(t, x, y_1, y_2, \ldots, y_{n-1}) \end{aligned}$$

of n coupled first order differential equations.

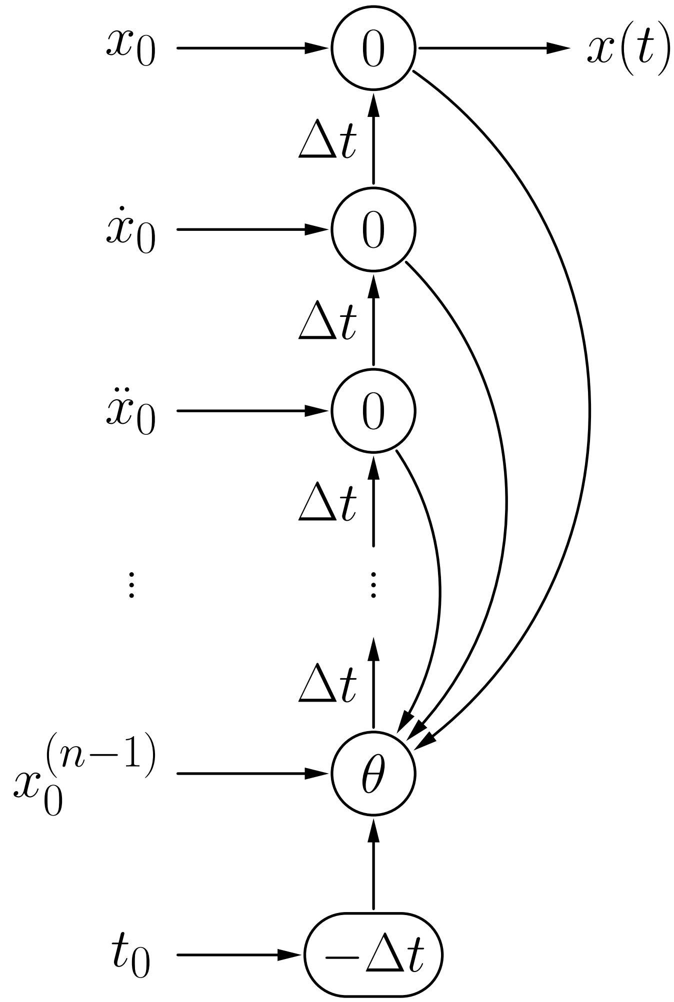
Recurrent Networks: Differential Equations

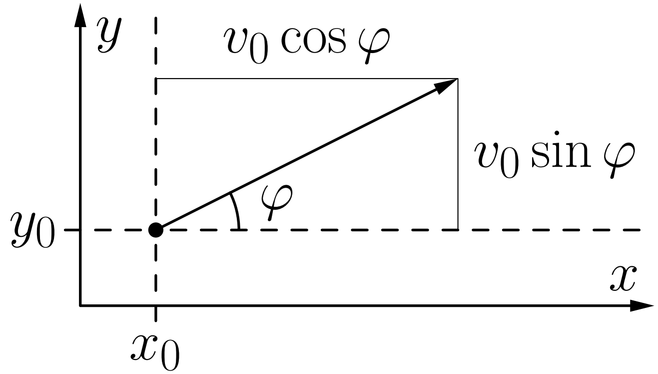
Replace differential quotient by forward distance to obtain the recursive equations

$$
\begin{aligned}
x(t_i) &= x(t_{i-1}) + \Delta t \cdot y_1(t_{i-1}), \\
y_1(t_i) &= y_1(t_{i-1}) + \Delta t \cdot y_2(t_{i-1}), \\
&\vdots \\
y_{n-2}(t_i) &= y_{n-2}(t_{i-1}) + \Delta t \cdot y_{n-3}(t_{i-1}), \\
y_{n-1}(t_i) &= y_{n-1}(t_{i-1}) + f(t_{i-1}, x(t_{i-1}), y_1(t_{i-1}), \ldots, y_{n-1}(t_{i-1}))
\end{aligned}
$$

- Each of these equations describes the update of one neuron.
- The last neuron needs a special activation function.

Recurrent Networks: Differential Equations





Diagonal throw of a body.

Two differential equations (one for each coordinate):

$$\ddot{x} = 0 \qquad \text{and} \qquad \ddot{y} = -g,$$

where $g = 9.81\,\mathrm{ms}^{-2}$.

Initial conditions $x(t_0) = x_0$, $y(t_0) = y_0$, $\dot{x}(t_0) = v_0 \cos\varphi$ and $\dot{y}(t_0) = v_0 \sin\varphi$.

Recurrent Networks: Diagonal Throw

Introduce intermediary quantities

$$v_x = \dot{x} \qquad \text{and} \qquad v_y = \dot{y}$$

to reach the system of differential equations:

$$\begin{aligned} \dot{x} &= v_x, & \qquad \dot{v}_x &= 0, \\ \dot{y} &= v_y, & \qquad \dot{v}_y &= -g, \end{aligned}$$

from which we get the system of recursive update formulae

$$\begin{aligned} x(t_i) &= x(t_{i-1}) + \Delta t\ v_x(t_{i-1}), & \qquad v_x(t_i) &= v_x(t_{i-1}), \\ y(t_i) &= y(t_{i-1}) + \Delta t\ v_y(t_{i-1}), & \qquad v_y(t_i) &= v_y(t_{i-1}) - \Delta t\ g. \end{aligned}$$

Recurrent Networks: Diagonal Throw

Better description: Use **vectors** as inputs and outputs

$$\ddot{\vec{r}} = -g\vec{e}_y,$$

where $\vec{e}_y = (0, 1)$.

Initial conditions are $\vec{r}(t_0) = \vec{r}_0 = (x_0, y_0)$ and $\dot{\vec{r}}(t_0) = \vec{v}_0 = (v_0 \cos\varphi, v_0 \sin\varphi)$.

Introduce one **vector-valued** intermediary quantity $\vec{v} = \dot{\vec{r}}$ to obtain

$$\dot{\vec{r}} = \vec{v}, \qquad \dot{\vec{v}} = -g\vec{e}_y$$

This leads to the recursive update rules

$$\begin{aligned}
\vec{r}(t_i) &= \vec{r}(t_{i-1}) + \Delta t\ \vec{v}(t_{i-1}), \\
\vec{v}(t_i) &= \vec{v}(t_{i-1}) - \Delta t\ g\vec{e}_y
\end{aligned}$$

Advantage of vector networks becomes obvious if friction is taken into account:

$$\vec{a} = -\beta\vec{v} = -\beta\dot{\vec{r}}$$

β is a constant that depends on the size and the shape of the body.
This leads to the differential equation

$$\ddot{\vec{r}} = -\beta\dot{\vec{r}} - g\vec{e}_y.$$

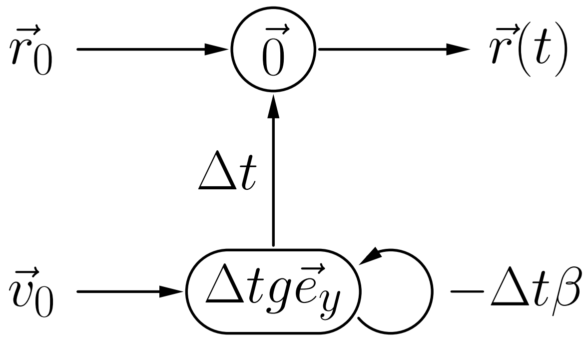
Introduce the intermediary quantity $\vec{v} = \dot{\vec{r}}$ to obtain

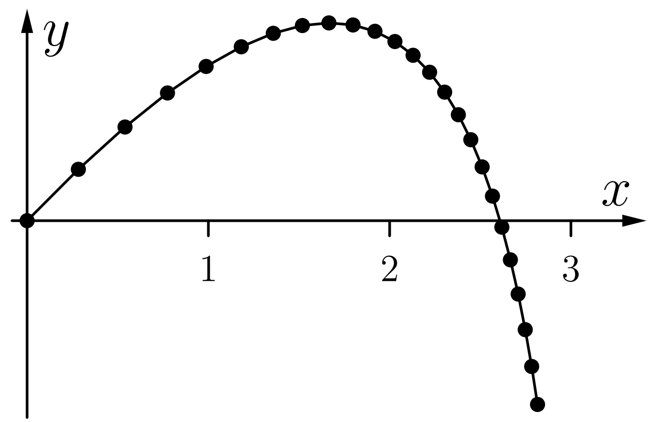
$$\dot{\vec{r}} = \vec{v}, \qquad \dot{\vec{v}} = -\beta\vec{v} - g\vec{e}_y,$$

from which we obtain the recursive update formulae

$$\begin{aligned}\vec{r}(t_i) &= \vec{r}(t_{i-1}) + \Delta t\ \vec{v}(t_{i-1}),\\ \vec{v}(t_i) &= \vec{v}(t_{i-1}) - \Delta t\ \beta\ \vec{v}(t_{i-1}) - \Delta t\ g\vec{e}_y.\end{aligned}$$

Resulting recurrent neural network:



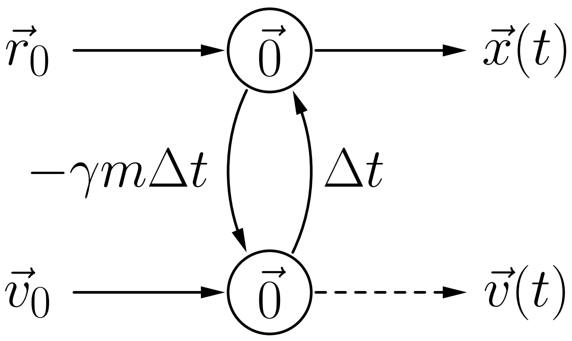


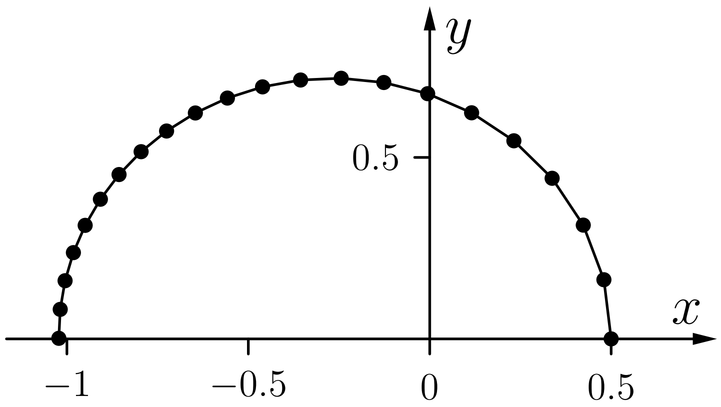
- There are no strange couplings as there would be in a non-vector network.
- Note the deviation from a parabola that is due to the friction.

$$\ddot{\vec{r}} = -\gamma m \frac{\vec{r}}{|\vec{r}|^3}, \qquad \Rightarrow \qquad \dot{\vec{r}} = \vec{v}, \qquad \dot{\vec{v}} = -\gamma m \frac{\vec{r}}{|\vec{r}|^3}.$$

Recursive update rules:

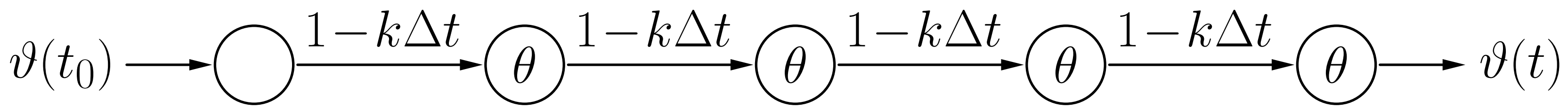
$$\begin{aligned} \vec{r}(t_i) &= \vec{r}(t_{i-1}) + \Delta t\ \vec{v}(t_{i-1}), \\ \vec{v}(t_i) &= \vec{v}(t_{i-1}) - \Delta t\ \gamma m \frac{\vec{r}(t_{i-1})}{|\vec{r}(t_{i-1})|^3}, \end{aligned}$$





Idea: Unfold the network between training patterns,
i.e., create one neuron for each point in time.

Example: **Newton's cooling law**



Unfolding into four steps. It is $\theta = -k\vartheta_A \Delta t$.

- Training is standard backpropagation on unfolded network.
- All updates refer to the same weight.
- updates are carried out after first neuron is reached.