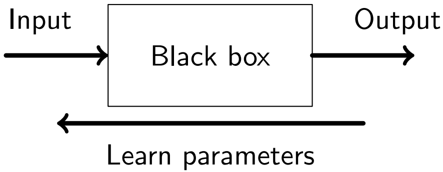
Scientific Machine Learning

Stephan Scholz

May 31, 2022

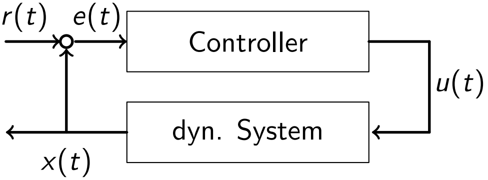
HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

(Scientific) Machine Learning



- Machine Learning:
  adjust parameters p to minimize $\textit{error} = \textit{output} - \textit{input}$
  $\Rightarrow$ Black box model
- System identification:
  find parameters p of dynamical system $\dot{x}(t) = f(x, t, p)$
  such that it fits to my experimental data $\tilde{x}(t) \approx x(t)$.
  $\Rightarrow$ Grey box model - system is partially known

(Scientific) Machine Learning

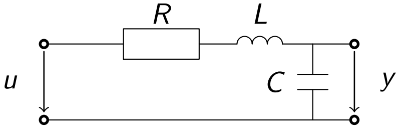


input u
states x
error e
reference r

- Control Engineering:
  find a control signal $u(t) = g(t, p)$ such that
  1. the dynamical systems "does not explode" (stability) and
  2. the error $e(t) = r(t) - x(t)$ is minimized

  $\Rightarrow$ Grey box model

Example: RLC circuit



- Ordinary Differential Equation

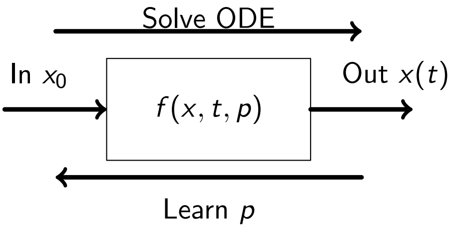
$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \frac{-1}{LC} & -\frac{RC}{LC} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{LC} \end{pmatrix} u(t)$$

$$y(t) = x_1(t)$$

- Resistor R
- Inductor L
- Capacitor C

System Identification

1. Create original data
2. Build grey box model (ODE with parameters)
3. Design cost (or goal) function
4. Choose initial parameters p and optimizer
5. Run optimization loop
   - Solve ODE
   - Apply continuous backpropagation
   - Adapt parameters



System Identification: RLC Circuit

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \frac{-1}{LC} & -\frac{R}{L} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{LC} \end{pmatrix} u(t)$$

1. Which parameters shall be learned?

System Identification: RLC Circuit

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \frac{-1}{LC} & -\frac{R}{L} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{LC} \end{pmatrix} u(t)$$

1. Which parameters shall be learned?
   - $p_1 = R$, $p_2 = L$ and $p_3 = C$

System Identification: RLC Circuit

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \frac{-1}{LC} & -\frac{R}{L} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{LC} \end{pmatrix} u(t)$$

1. Which parameters shall be learned?
    - $p_1 = R$, $p_2 = L$ and $p_3 = C$
    - $p_1 = \frac{1}{LC}$ and $p_2 = \frac{R}{L}$
2. Cost function

System Identification: RLC Circuit

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \frac{-1}{LC} & -\frac{R}{L} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{LC} \end{pmatrix} u(t)$$

1. Which parameters shall be learned?
   - $p_1 = R$, $p_2 = L$ and $p_3 = C$
   - $p_1 = \frac{1}{LC}$ and $p_2 = \frac{R}{L}$
2. Cost function

$$\min_p J(x, \tilde{x}) = \int_0^T (x_1(t) - \tilde{x}_1(t))^2 + (x_2(t) - \tilde{x}_2(t))^2 \, \mathrm{d}\, t$$

System Identification: RLC Circuit

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \frac{-1}{LC} & -\frac{R}{L} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{LC} \end{pmatrix} u(t)$$

1. Which parameters shall be learned?
   - $p_1 = R$, $p_2 = L$ and $p_3 = C$
   - $p_1 = \frac{1}{LC}$ and $p_2 = \frac{R}{L}$
2. Cost function

$$\min_p J(x, \tilde{x}) = \int_0^T (x_1(t) - \tilde{x}_1(t))^2 + (x_2(t) - \tilde{x}_2(t))^2 \,\mathrm{d}\, t$$

3. Optimizer
   - local vs. global
   - convex vs. non-convex
   - here: BFGS

System Identification: RLC Circuit

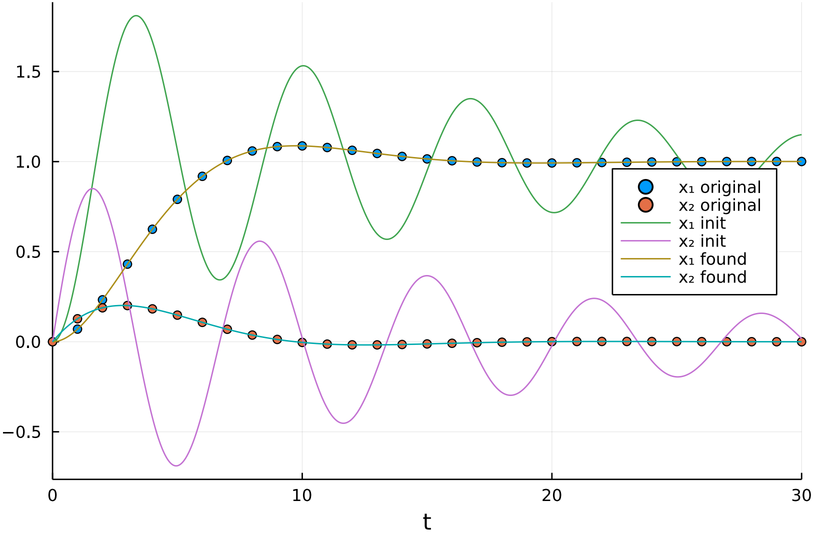
$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \frac{-1}{LC} & -\frac{R}{L} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{LC} \end{pmatrix} u(t)$$

1. Which parameters shall be learned?
   - $p_1 = R$, $p_2 = L$ and $p_3 = C$
   - $p_1 = \frac{1}{LC}$ and $p_2 = \frac{R}{L}$
2. Cost function

$$\min_p J(x, \tilde{x}) = \int_0^T (x_1(t) - \tilde{x}_1(t))^2 + (x_2(t) - \tilde{x}_2(t))^2 \, \mathrm{d}\, t$$

3. Optimizer
   - local vs. global
   - convex vs. non-convex
   - here: BFGS

Is it possible to replace parts of the ODE by ANNs?



1.5
1.0
0.5
0.0
−0.5
0
10
20
30
t
x₁ original
x₂ original
x₁ init
x₂ init
x₁ found
x₂ found

Neural Ordinary Differential Equations

Typical neural network architecture (e.g. ResNets) can be represented

$$x(n+1) = x(n) + h\ f(x(n), p) \tag{1}$$

with $x(n)$ the states of the recent layer and p the set of parameters.

Initial Question
How is it possible to accelerate deep learning with such architectures?

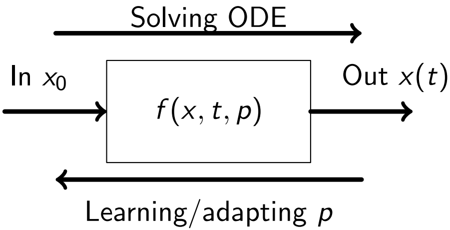
Idea
Equation (1) looks like the Euler forward method. Let's go back to the ODE and solve it with a better method (e.g. Runge-Kutta)!

Scheme

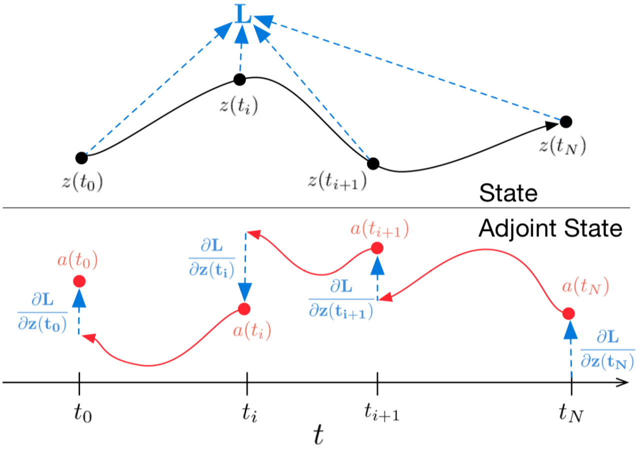
- Go backwards from Euler approximation to (Neural) ODE

$$\frac{x(n+1) - x(n)}{h} = f(x(n), p) \quad \Rightarrow \quad \dot{x}(t) = f(x, t, p)$$

- Solve Neural ODE and learn/adapt parameters p
- Optimization with standard tools (e.g. BFGS) or ML tools (e.g. ADAM)



Continuous Backpropagation



from Chen, et al.: Neural ordinary differential equations [1]

Two Perspectives on Neural ODE

Machine Learning

- Build "infinitely" deep neural networks
- Works also for other ML tools like *Normalizing Flows*
- Neural ODE must fit to "mathematical rules"

Two Perspectives on Neural ODE

Machine Learning

- Build “infinitely” deep neural networks
- Works also for other ML tools like *Normalizing Flows*
- Neural ODE must fit to “mathematical rules”

Modeling, Simulation and Control

- Integration of ML tools (ANN, optimizers,) in differential equations
- Also implemented for DAE, SDE, etc.
- Usually not *quick-and-dirty*

Optimal Control of RLC circuit with ANN

$$\min_p \int_0^T (r(t) - y(t))^2 dt$$

subject to

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \frac{-1}{LC} & -\frac{R}{L} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{LC} \end{pmatrix} u(t)$$

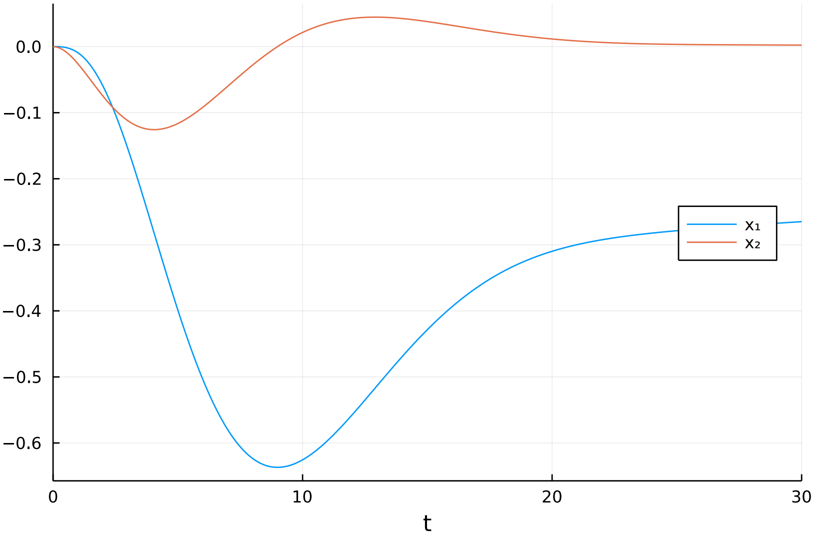
$$y(t) = x_1(t)$$

$$u(t) = ANN(t, p)$$

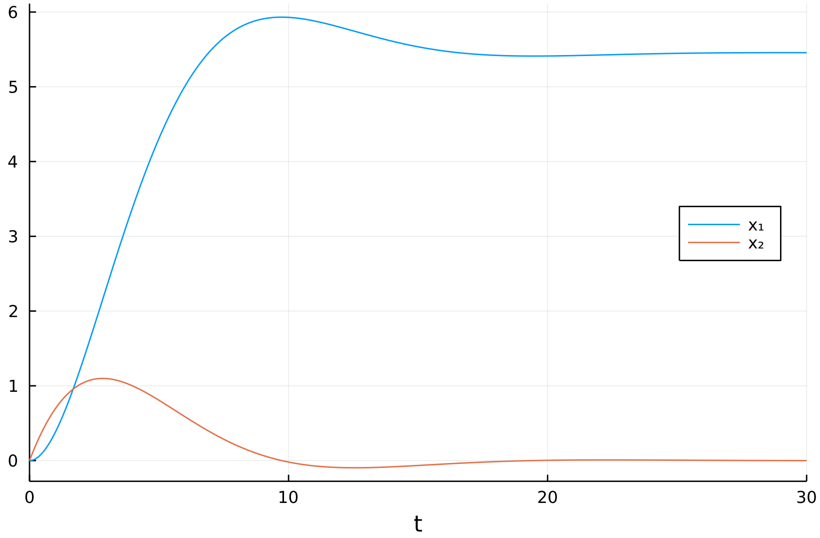
We assume

- coefficients $R = 1,\ L = 2,\ C = 3$ and
- reference $r(t) = 5$
- 4-layer network, 32 neurons per hidden layer: 1153 parameters

Optimal Control: initial parameters



Optimal Control: trained parameters



What else?

The Magic

- Julia environment + SciML Organization
- High-class numerical integrators
- Large range of optimizers
- *Automatic Differentiation*

What else?

The Magic

- Julia environment + SciML Organization
- High-class numerical integrators
- Large range of optimizers
- *Automatic Differentiation*

The Universe

- Dynamic Mode Decomposition + SINDy
- Physics-Informed Neural Networks
- Many applications in biology, chemistry, physics, engineering...

Bibliography

- Ricky Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud: *Neural ordinary differential equations*. Advances in neural information processing systems (2018).

- Patrick Kidger: *On Neural Differential Equations*. arXiv preprint (2022). arXiv:2202.02435.

- Christopher Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, Vaibhav Dixit: *Diffeqflux.jl - A julia library for neural differential equations*. arXiv preprint (2019). arXiv:1902.02376.

- Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, Alan Edelman: *Universal differential equations for scientific machine learning*. arXiv preprint (2020). arXiv:2001.04385.

- Steven A. Frank: *Optimizing differential equations to fit data and predict outcomes*. arXiv preprint (2022). arXiv:2204.07833.