Numerical deep learning: Introduction to neural ordinary differential equations

Rafał Osadnik, Lorenz Ehrlich

28/01/2025

Ordinary differential equations

- **Differential equation** with single independent variable
- Equations describe an **aspect** of a system
- Derivatives describe the **dynamics**

$$x(t) = \textbf{position}$$

$$\frac{dx(t)}{dt} = \textbf{velocity} = x'(t)$$

$$\frac{d^2x(t)}{dt} = \textbf{acceleration} = x''(t)$$

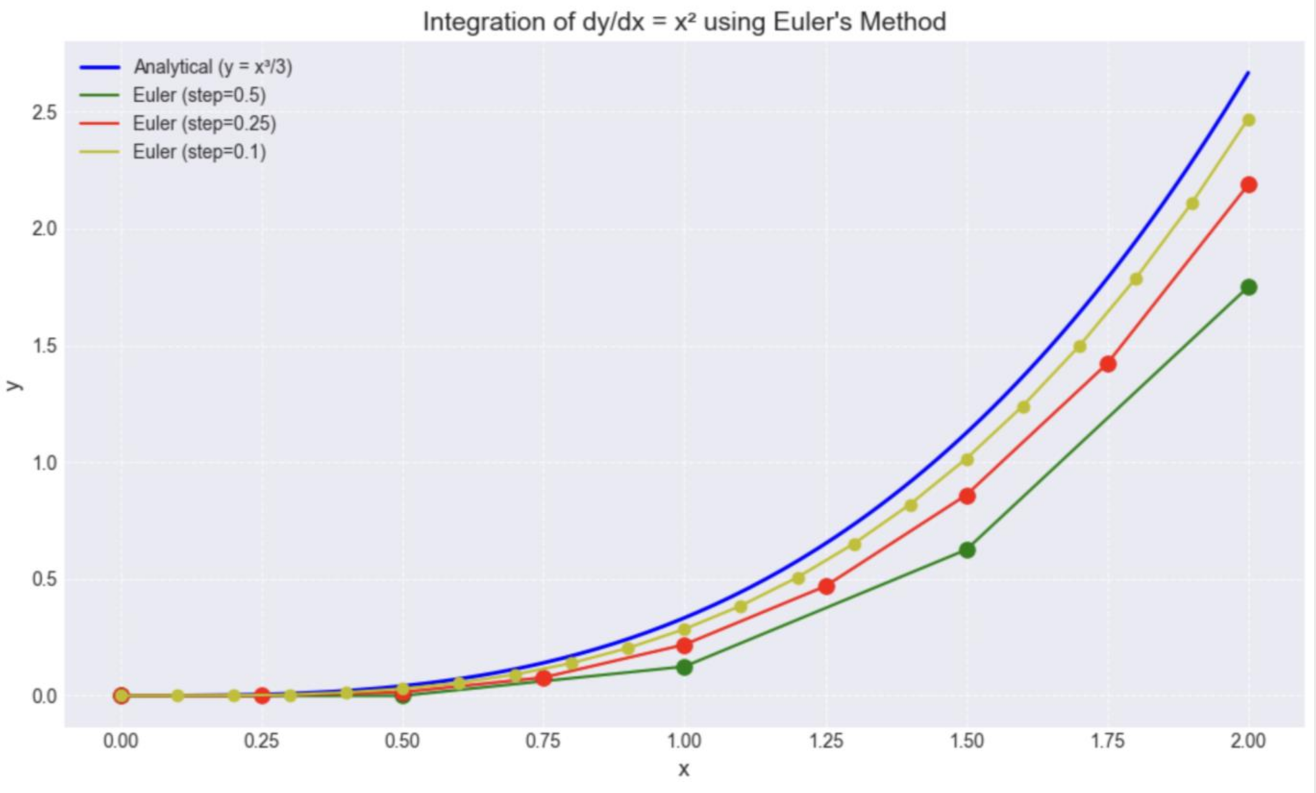
Numerical ODE solvers

Some ODEs are **impossible** to solve analytically. Numerical ODE solvers can **approximate** the solution.

Euler's method

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

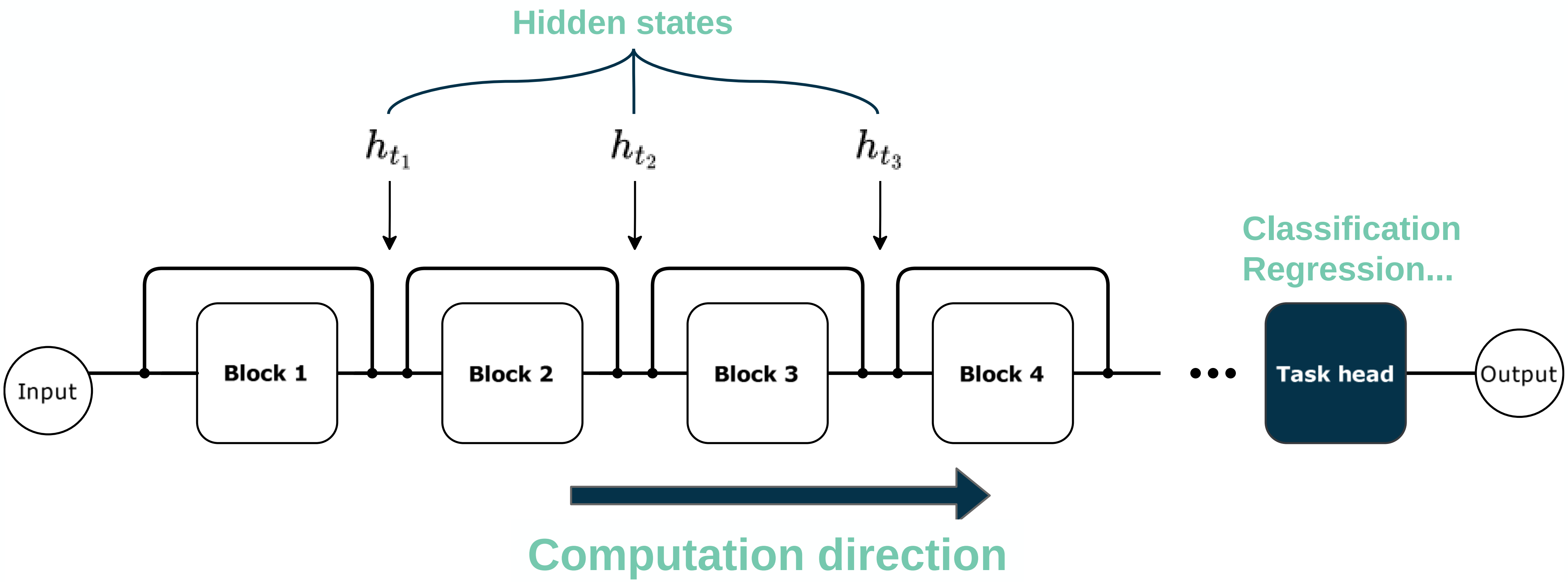
$$y'(t) = f\big(t, y(t)\big)$$



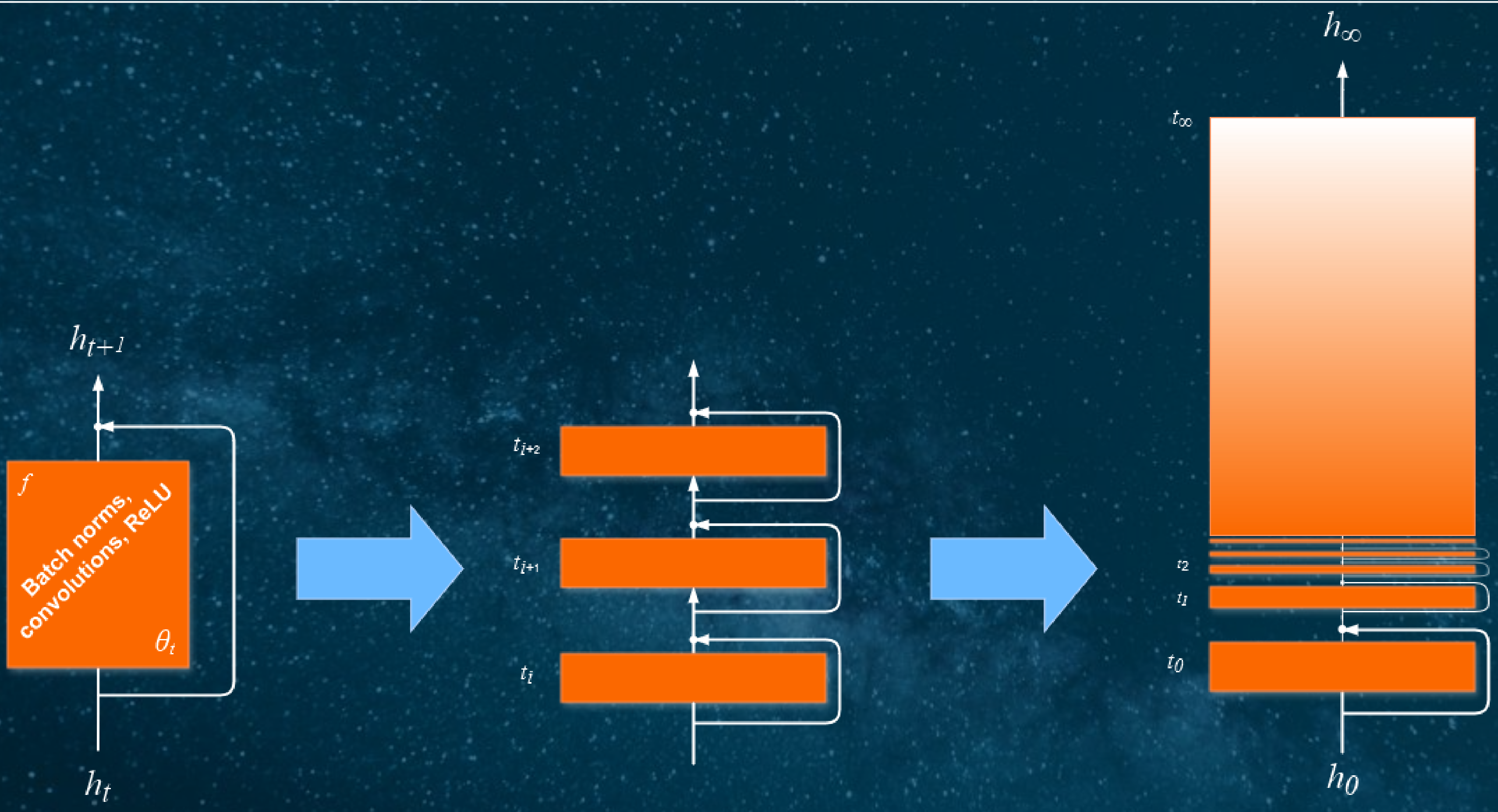
Parameters:
- Step size
- Initial condition

Neural ordinary differential equations (N-ODEs)

Hidden state of a neural network is the representation the network learns from the provided data.



N-ODEs – Infinite depth residual network



$$h_{t+1} = h_t + f(h_t, \theta_t)$$

$$h_N = h_0 + \sum_{t=0}^{N} f(h_t, \theta_t)$$

$$h(t_\infty) = h(t_0) + \int_{t_0}^{t_\infty} f\big(h(t), t, \theta\big)\, dt$$

N-ODEs – Dynamics of the hidden state

N-ODEs are based on learning the dynamics of the hidden state, not the hidden state itself. This enables learning a continuous description of the hidden state with arbitrarily chosen evaluation points.

$$h(t_\infty) = h(t_0) + \int_{t_0}^{t_\infty} f\big(h(t), t, \theta\big)\, dt \quad \Rightarrow \quad f = h'$$

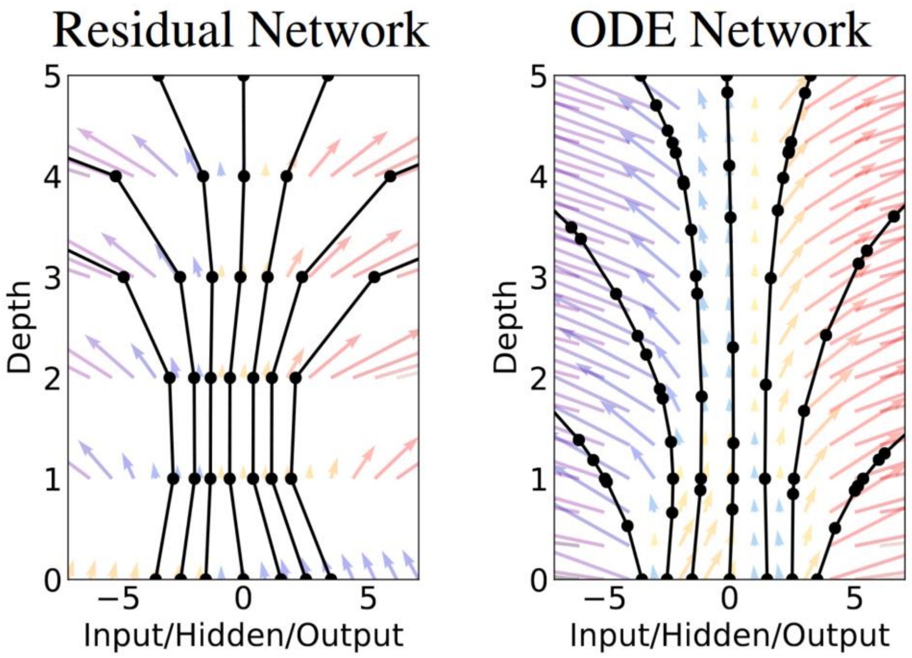
Neural network

Inference with N-ODEs

- 3 step process
  - Encode to hidden state
  - Compute over time interval
  - Decode to output state

$$h(t_1) = h(t_0) + \int_{t_0}^{t_1} f(h(t), t, \theta)dt = \text{ODESolve}\Big(h(t_0), f, t_0, t_1, \theta\Big)$$

The hidden state is the solution at some chosen time T.



Initial condition / input

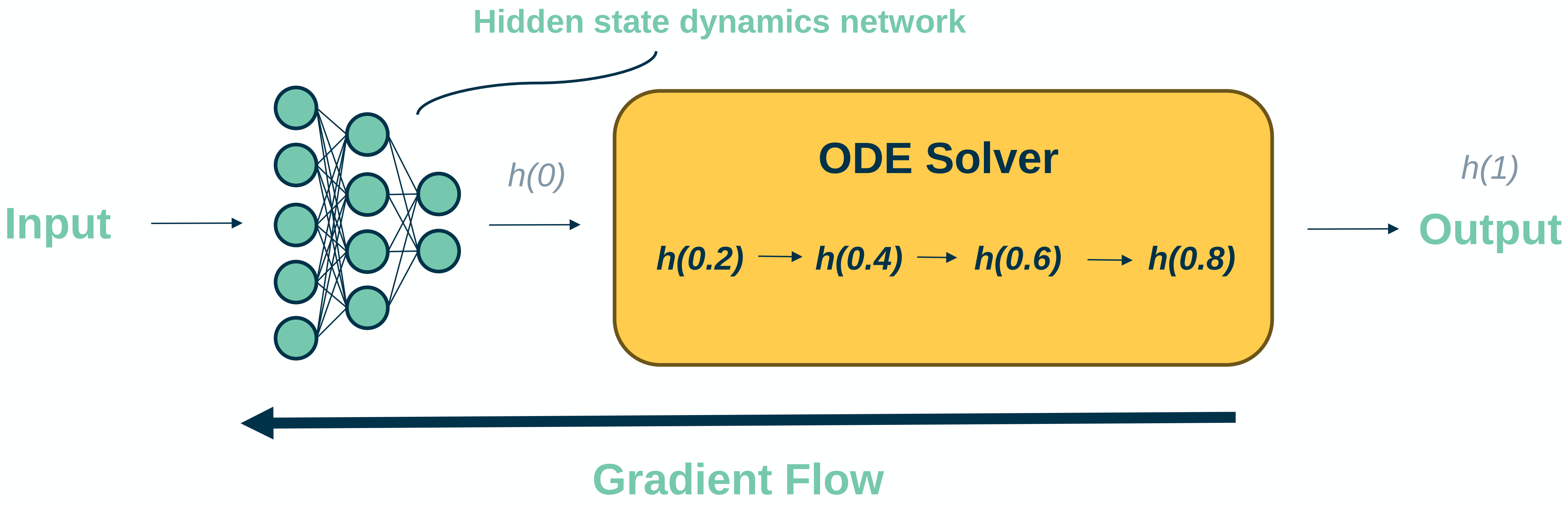
$$h(T) = h(t_0) + \int_{t_0}^{T} f\big(h(t), t, \theta\big)dt$$

Hidden state

Hidden state dynamics

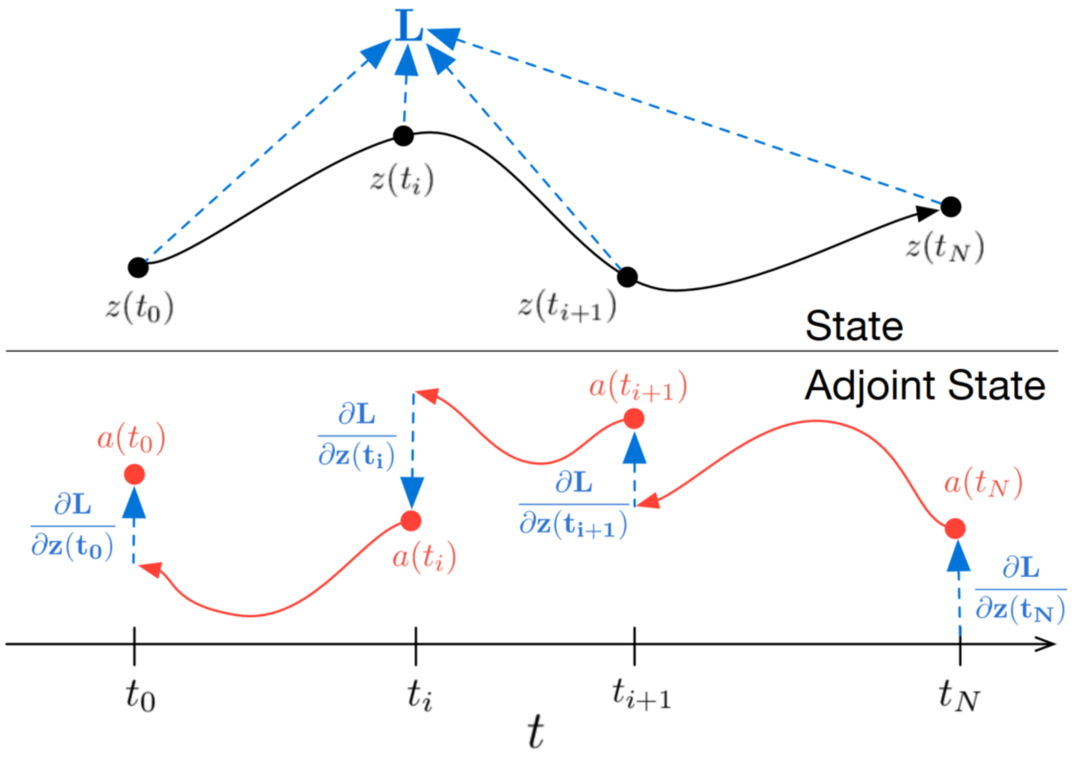
N-ODEs training - Backpropagation

Simplest learning can be done through **backpropagation**, given the used solver is **differentiable**. This is memory inefficient.



N-ODEs training – Adjoint sensitivity

There is an alternative – the **adjoint sensitivity method**. It treats the ODE solver as a black box, and calcualtes gradients using the adjoint state. Instead of backpropagating, we can solve **additional ODEs to get gradients**.



Reverse mode differentiation of the hidden state ODE [1]

Solve 3 ODEs:

- Forward in time to get the hidden state(s)
- Backwards in time to get the adjoint states
- Backwards in time to get the gradients

DEMOS

Bibliography

[1]. **Neural Ordinary Differential Equations,** Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud.