ResNet after all?

How (not) to design *continuous* neural network architectures

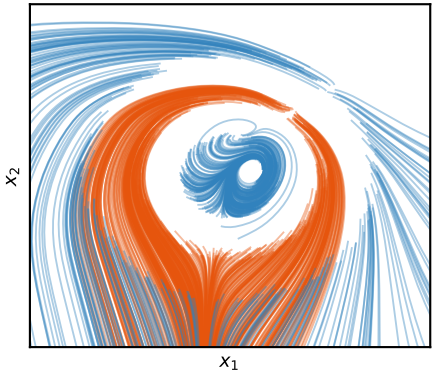
Katharina Ott

Continuous Time Perspectives in Machine Learning Workshop

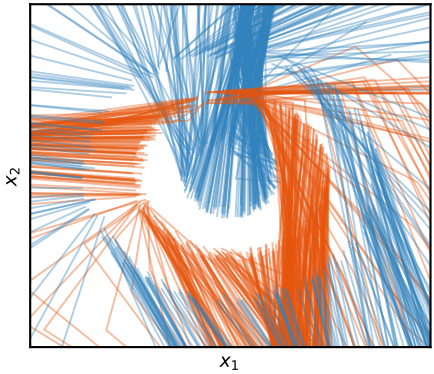
23 July 2022

Overview

Training #steps = 256



Training #steps = 2



Katharina Ott

Introduction
Introduction to ODEs

$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = f(z(t), t), \quad z(t = t_0) = z_0, \quad t \in [t_0, T]$$

$f : \mathbb{R}^n \to \mathbb{R}^n$ - some function (vector field)

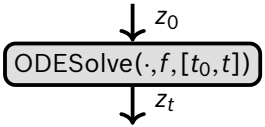
$$z(t) = z_0 + \int_{t_0}^{t} f(z(s), s)\mathrm{d}s$$

Katharina Ott

Introduction
Introduction to ODEs

$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = f(z(t), t), \quad z(t = t_0) = z_0, \quad t \in [t_0, T]$$

$f : \mathbb{R}^n \to \mathbb{R}^n$ - some function (vector field)



$$z(t) = z_0 + \int_{t_0}^{t} f(z(s), s)\mathrm{d}s \approx \mathrm{ODESolve}(z, f, [t_0, t])$$

Introduction
Introduction to ODEs

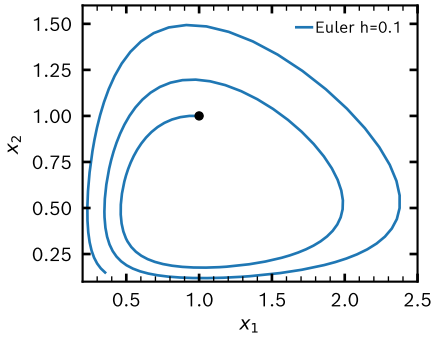
$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = f(z(t), t), \quad z(t = t_0) = z_0, \quad t \in [t_0, T]$$

$f : \mathbb{R}^n \to \mathbb{R}^n$ - some function (vector field)

$$z(t) = z_0 + \int_{t_0}^{t} f(z(s), s)\mathrm{d}s = \mathrm{ODESolve}(z, f, [t_0, t])$$

Numerical scheme to solve the ODE
e.g. Euler's method $t_{k+1} = t_k + h$

$$z(t_{k+1}) = z(t_k) + h f_\theta(z(t_k), t_k)$$



$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = \alpha x_1 - \beta x_1 x_2$$

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = \delta x_1 x_2 - \gamma x_2$$

Katharina Ott

Introduction
Introduction to ODEs

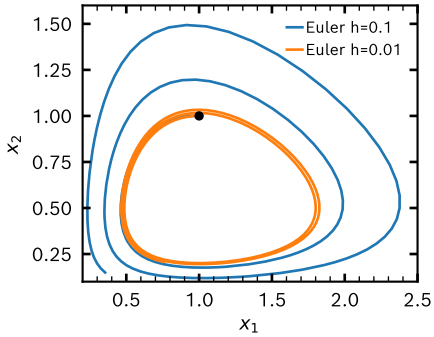
$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = f(z(t), t), \quad z(t = t_0) = z_0, \quad t \in [t_0, T]$$

$f : \mathbb{R}^n \to \mathbb{R}^n$ - some function (vector field)

$$z(t) = z_0 + \int_{t_0}^{t} f(z(s), s)\mathrm{d}s = \mathrm{ODESolve}(z, f, [t_0, t])$$

Numerical scheme to solve the ODE
e.g. Euler's method $t_{k+1} = t_k + h$

$$z(t_{k+1}) = z(t_k) + h f_\theta(z(t_k), t_k)$$



$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = \alpha x_1 - \beta x_1 x_2$$

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = \delta x_1 x_2 - \gamma x_2$$

Katharina Ott

Introduction
Introduction to ODEs

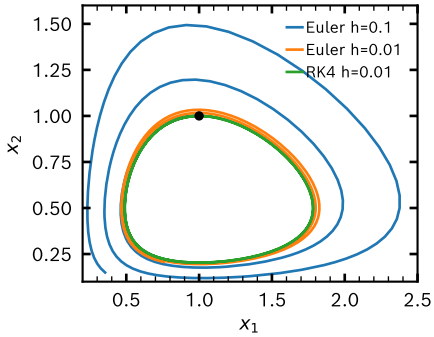
$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = f(z(t), t), \quad z(t = t_0) = z_0, \quad t \in [t_0, T]$$

$f : \mathbb{R}^n \to \mathbb{R}^n$ - some function (vector field)

$$z(t) = z_0 + \int_{t_0}^{t} f(z(s), s)\mathrm{d}s = \mathrm{ODESolve}(z, f, [t_0, t])$$

Numerical scheme to solve the ODE
e.g. Euler's method $t_{k+1} = t_k + h$

$$z(t_{k+1}) = z(t_k) + h f_\theta(z(t_k), t_k)$$



$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = \alpha x_1 - \beta x_1 x_2$$

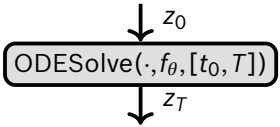
$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = \delta x_1 x_2 - \gamma x_2$$

Katharina Ott

Introduction
Introduction to *neural* ODEs

$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = f_\theta(z(t), t), \quad z(t = t_0) = z_0, \quad t \in [t_0, T]$$

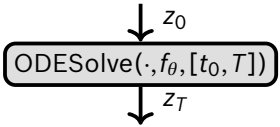
$f : \mathbb{R}^n \to \mathbb{R}^n$ - some *neural network* (vector field)



Katharina Ott

Introduction
Introduction to *neural* ODEs

$$\frac{dz(t)}{dt} = f_\theta(z(t), t), \quad z(t = t_0) = z_0, \quad t \in [t_0, T]$$



$f : \mathbb{R}^n \to \mathbb{R}^n$ - some *neural network* (vector field)

$$f_w = ... f_d \circ \text{ODESolve}(\cdot, f_\theta, [t_0, T]) \circ f_u ...$$

Loss: Batch data $\mathcal{B}$

$$L(w) = \frac{1}{|\mathcal{B}|} \sum_{\{z,y\} \in \mathcal{B}} l(y, f_w(z))$$

Classification Tasks
ResNets

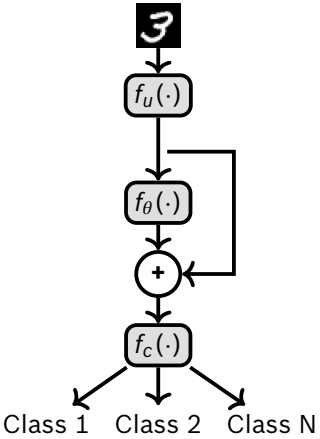
Solving Neural ODEs with Euler's method

$$z(t_{k+1}) = z(t_k) + hf_\theta(z(t_k))$$

Note similarity to ResNets ($h = 1$)

$$z_1 = z_0 + f_\theta(z_0)$$

→ Why not use Neural ODEs for classification tasks?



Classification Tasks
ResNets

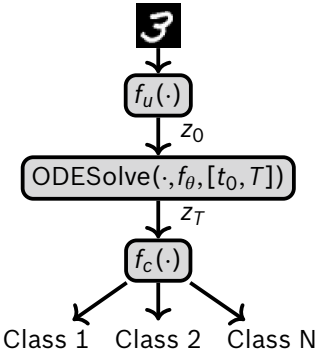
Solving Neural ODEs with Euler's method

$$z(t_{k+1}) = z(t_k) + hf_\theta(z(t_k))$$

Note similarity to ResNets ($h = 1$)

$$z_1 = z_0 + f_\theta(z_0)$$

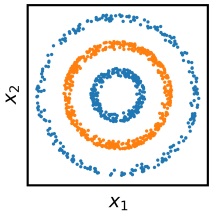
→ Why not use Neural ODEs for classification tasks?

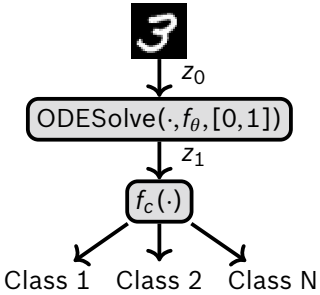


Katharina Ott

Classification Tasks
Classification tasks

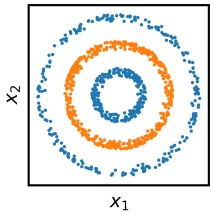
- Keep architecture simple to maximize the effect of the ODE-block
- Train with different solvers and different step sizes



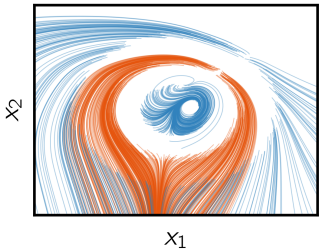


Katharina Ott

Neural ODEs and Their Solutions
Discrete dynamics

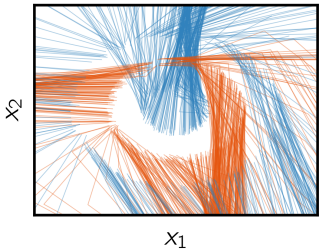


Training #steps = 256



$$\text{ODESolve}(z_0, f_\theta, [0,1], \text{solver} = \text{Euler}, \text{step size} = 1/256)$$

Training #steps = 2



$$\text{ODESolve}(z_0, f_\theta, [0,1], \text{solver} = \text{Euler}, \text{step size} = 1/2)$$

Katharina Ott

Neural ODEs and Their Solutions
Simple example

Training

$\text{ODESolve}(z_0, f_\theta, [0,1], \text{solver} = \text{train solver}, \text{step size} = \text{train step size})$

Train model $\rightarrow \ \theta^*$

Testing

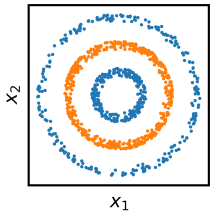
$\text{ODESolve}(z_0, f_{\theta^*}, [0,1], \text{solver} = \text{test solver}, \text{step size} = \text{test step size})$

Settings (for now):

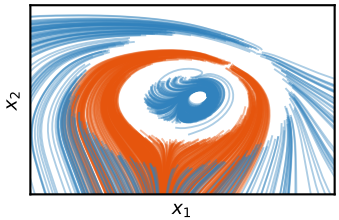
- test solver = train solver = Euler
- vary test step size

Katharina Ott

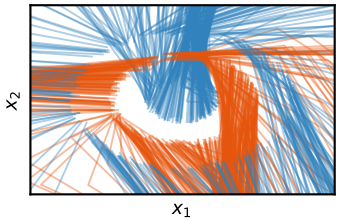
Neural ODEs and Their Solutions
Discrete dynamics



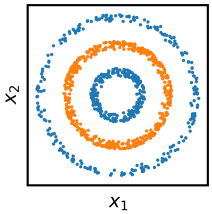
Training #steps = 256

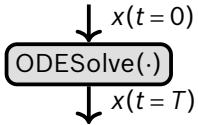


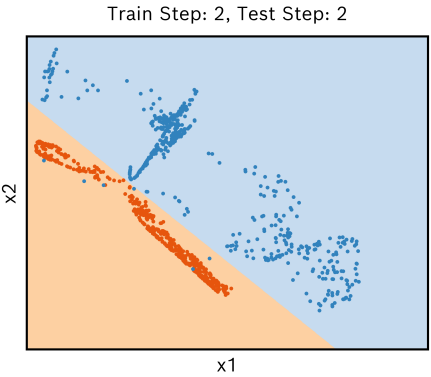
Training #steps = 2

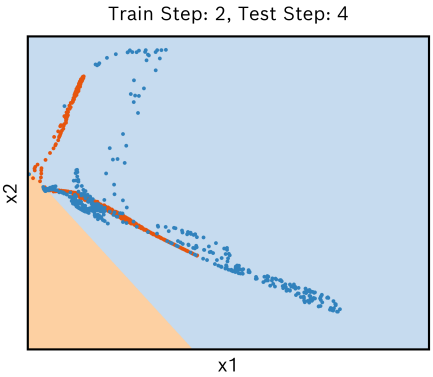


Neural ODEs and Their Solutions
Discrete dynamics







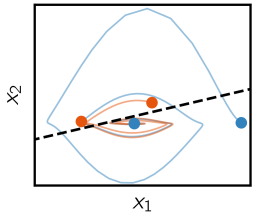


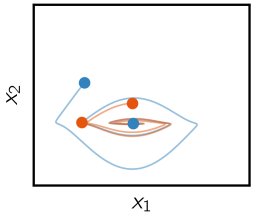
Katharina Ott

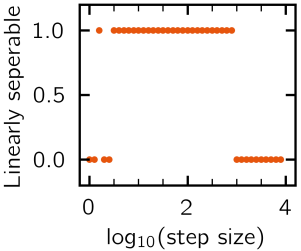
Neural ODEs and Their Solutions
Lady Windemere's Fan

Toy example:

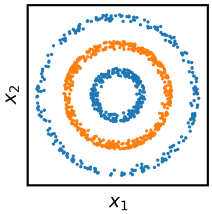
- Toy problem - XOR problem
  - $\{0,0\} \to 0$, $\{0,1\} \to 1$, $\{1,0\} \to 1$, $\{1,1\} \to 0$
- Dynamics describe moving along concentric ellipses with increasing velocity

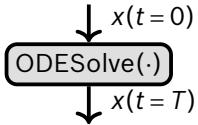


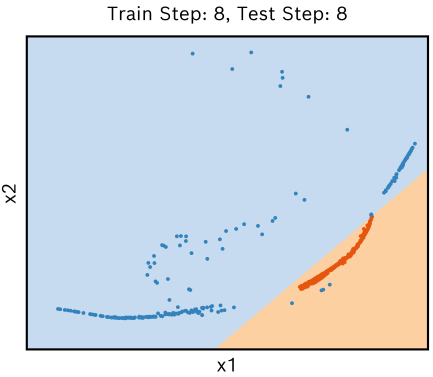


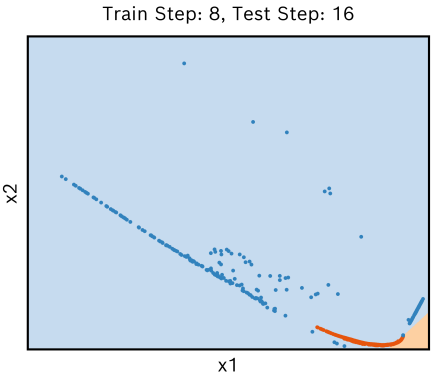


Neural ODEs and Their Solutions
Lady Windemere's Fan

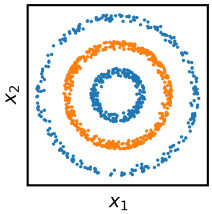


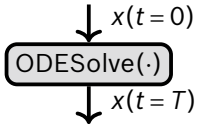




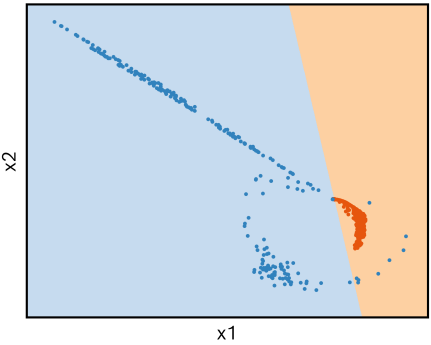


Neural ODEs and Their Solutions
Lady Windemere's Fan

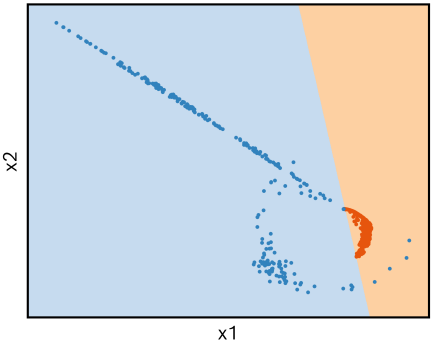




Train Step: 128, Test Step: 128



Train Step: 128, Test Step: 256



Katharina Ott

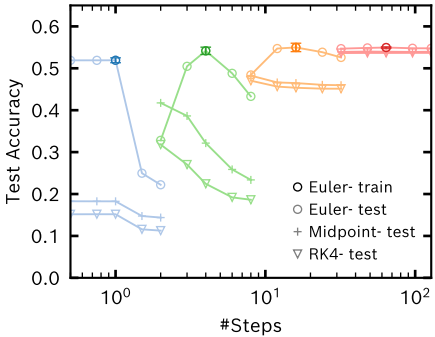
Neural ODEs and Their Solutions
Effects

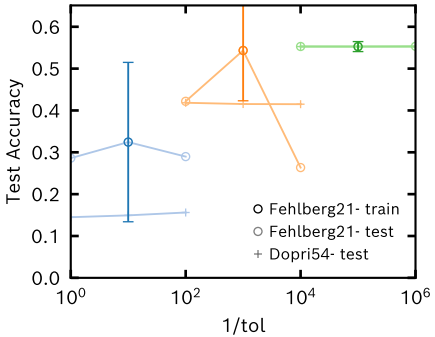
Effects that lead to drop in performance

- Trajectory crossing ↯ODEness
- Accumulation of error
  - For $h_{\text{test}} \to 0$ the solution does no longer remain linearly separable
  - For $h_{\text{test}} \to 0$ samples cross the decision boundary

We cannot easily distinguish these effects!

Neural ODEs and Their Solutions
Results on CIFAR10

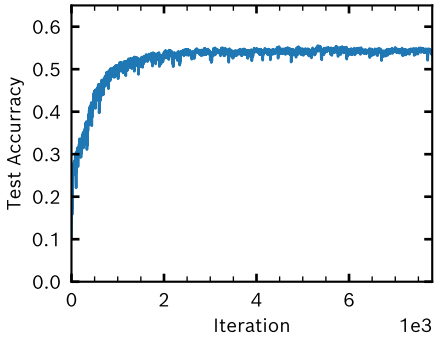


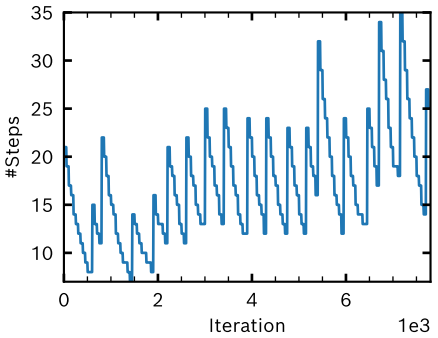


Proposed Solution
Step adaptation algorithm

```
Inputs ϵ, train_solver, test_solver, model;
while Training do
    ...
    train_solver_acc = model.calculate_acc(train_solver(ϵ));
    if Iteration % 50 == 0 then
        test_solver_acc = model.calculate_acc(test_solver(ϵ));
        if |train_solver_acc-test_solver_acc| > 0.1 then
            ϵ = 0.5 ϵ;
        else
            ϵ = 1.1 ϵ
        end
    end
end
```

Proposed Solution
Step adaptation algorithm

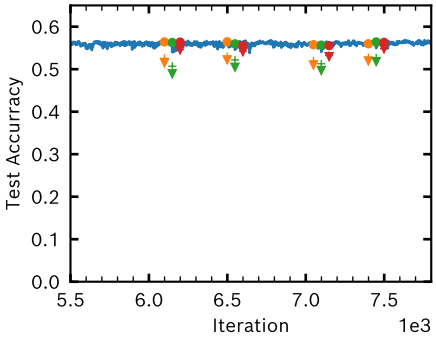


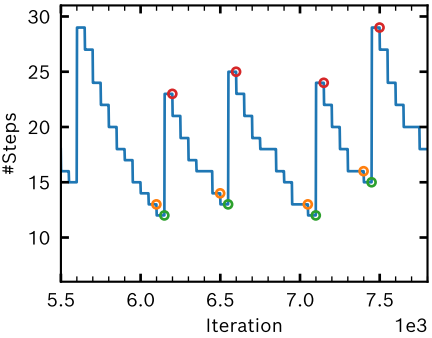


- Critical step size found by grid search: 17-33
- Average step size chosen by algorithm: 21.9

Katharina Ott

Proposed Solution
Step adaptation algorithm





Related Work
Results in other settings

- Testing neural ODE models with more precise solver:
  - Gusak et al. [2020] and Zhuang et al. [2020] on neural ODE architectures based on ResNet4, ResNet10 and ResNet18.
  - Queiruga et al. [2020] study for which ODE solvers ResNet models become continuous. Additionally, example on time series tasks.
- Achieving continuous dynamics in physical models [Krishnapriyan et al., 2022].
- Do ResNets become neural ODEs in the infinite depth limit [Cohen et al., 2021, Thorpe and van Gennip, 2018]?

Summary
Summary

- Step size impacts whether model maintains properties of ODE solutions
- Simple test: Test model with numerically more accurate solver
- Propose step adaptation algorithm

Katharina Ott

Prateek Katiyar

Philipp Hennig

Michael Tiemann

Link to paper: https://openreview.net/forum?id=HxzSxSxLOJZ
Link to slides: https://github.com/boschresearch/numerics_independent_neural_odes/tree/main/ICML2022Workshop

Paper

Slides

Julia Gusak, Larisa Markeeva, Talgat Daulbaev, Alexander Katrutsa, Andrzej Cichocki, and Ivan Oseledets. Towards understanding normalization in neural ODEs. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ODE. In *Proceedings of the 37th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2020.

Alejandro F Queiruga, N Benjamin Erichson, Dane Taylor, and Michael W Mahoney. Continuous-in-depth neural networks. *arXiv preprint arXiv:2008.02389*, 2020.

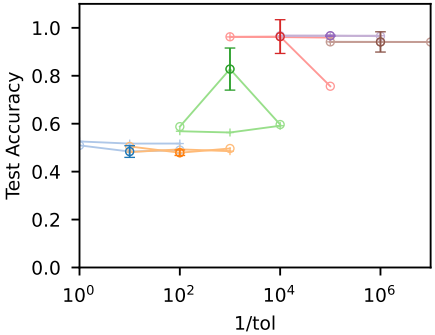
Aditi S Krishnapriyan, Alejandro F Queiruga, N Benjamin Erichson, and Michael W Mahoney. Learning continuous models for continuous physics. *arXiv preprint arXiv:2202.08494*, 2022.

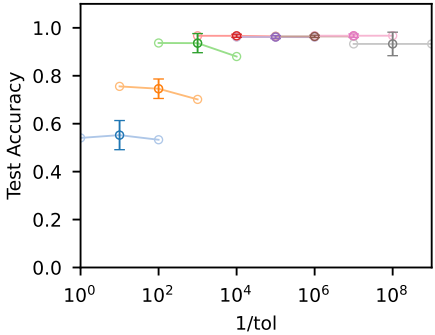
Alain-Sam Cohen, Rama Cont, Alain Rossier, and Renyuan Xu. Scaling properties of deep residual networks. In *Proceedings of the 38th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2021.

Matthew Thorpe and Yves van Gennip. Deep limits of residual neural networks. *arXiv preprint arXiv:1810.11741*, 2018.

Appendix
Adjoint training

Results for Sphere2

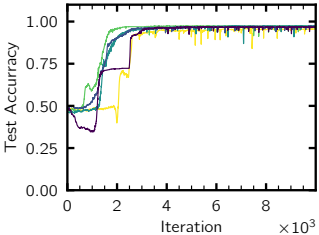


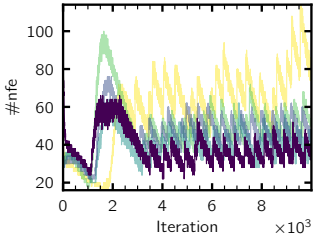


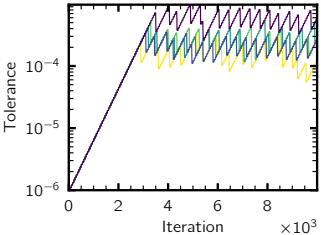
Katharina Ott

Appendix
Tolerance adaptation algorithm

Results for Sphere2 with Fehlberg2(1)







Katharina Ott

Appendix
Architecture Sphere2

Neural ODE Block

- Conv1D(1, 32, Kernel 1x1, padding 0) + ReLu
- Conv1D(32, 32, Kernel 3x3, padding 1) + ReLu
- Conv1D(32, 1, Kernel 1x1, padding 0)

Classifier

- Flatten + LinearLayer(2,2) + SoftMax

Hyper-parameters

- Batch size: 128
- Optimizer: Adam
- Learning rate: 1e-4
- Iterations used for training: 10000

Appendix
Architecture CIFAR10

Neural ODE Block

- Conv2D(3, 128, Kernel 1x1, padding 0) + ReLu
- Conv2D(128, 128, Kernel 3x3, padding 1) + ReLu
- Conv2D(128, 3, Kernel 1x1, padding 0)

Classifier

- Flatten + LinearLayer(3072,10) + SoftMax

Hyper-parameters

- Batch size: 256
- Optimizer: Adam
- Learning rate: 1e-3
- Iterations used for training: 7800