arXiv:2007.04504v2 [cs.LG] 22 Oct 2020

Learning differential equations that are easy to solve

Jacob Kelly*
University of Toronto, Vector Institute
jkelly@cs.toronto.edu

Jesse Bettencourt*
University of Toronto, Vector Institute
jessebett@cs.toronto.edu

Matthew James Johnson
Google Brain
mattjj@google.com

David Duvenaud
University of Toronto, Vector Institute
duvenaud@cs.toronto.edu

Abstract

Differential equations parameterized by neural networks become expensive to solve numerically as training progresses. We propose a remedy that encourages learned dynamics to be easier to solve. Specifically, we introduce a differentiable surrogate for the time cost of standard numerical solvers, using higher-order derivatives of solution trajectories. These derivatives are efficient to compute with Taylor-mode automatic differentiation. Optimizing this additional objective trades model performance against the time cost of solving the learned dynamics. We demonstrate our approach by training substantially faster, while nearly as accurate, models in supervised classification, density estimation, and time-series modelling tasks.

1 Introduction

Differential equations describe a system's behavior by specifying its instantaneous dynamics. Historically, differential equations have been derived from theory, such as Newtonian mechanics, Maxwell's equations, or epidemiological models of infectious disease, with parameters inferred from observations. Solutions to these equations usually cannot be expressed in closed-form, requiring numerical approximation.

Recently, ordinary differential equations parameterized by millions of learned parameters, called neural ODEs, have been fit for latent time series models, density models, or as a replacement for very deep neural networks (Rubanova et al., 2019; Grathwohl et al., 2019; Chen et al., 2018). These learned models are not constrained to match a theoretical model, only to optimize an objective on observed data. Learned models with nearly indistinguishable predictions can have substantially different dynamics. This raises the possibility that we can find equivalent models that are easier and faster to solve. Yet standard training methods have no way to penalize the complexity of

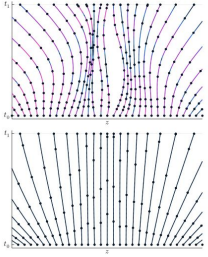


Figure 1: *Top:* Trajectories of an ODE fit to map $\mathbf{z}(t_1) = \mathbf{z}(t_0) + \mathbf{z}(t_0)^3$. The learned dynamics are unnecessarily complex and require many evaluations (black dots) to solve. *Bottom:* Regularizing the third total deriva-

Learning Differential Equations that are Easy to Solve

Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, David Duvenaud

Evgenii Egorov

Outline

1. Get intuition about: what do we want to affect?
2. Recalls: RK methods, step-size selection
3. =1 + 2: objective from paper

Motivation

Motivation $\dot{z} = z$

- Consider (some how) we train ODE: $\dot{z} = f_\theta(z, t)$
-
-
-

Motivation $\dot{z} = z$

- Consider (some how) we train ODE: $\dot{z} = f_\theta(z, t)$
- How do we apply it?
  By (**numerically**) solving IVP problem: $z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_\theta(z(t), t)\, dt$

Motivation $\dot{z} = z$

- Consider (some how) we train ODE: $\dot{z} = f_\theta(z, t)$
- How do we apply it?
  By (**numerically**) solving IVP problem: $z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_\theta(z(t), t)\, dt$
- Numerical Solver evaluates: $f_\theta(z(t), t)$
-

Motivation $\dot{z} = z$

- Consider (some how) we train ODE: $\dot{z} = f_\theta(z, t)$
- How do we apply it?
  By (**numerically**) solving IVP problem: $z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_\theta(z(t), t)\, dt$
- Numerical Solver evaluates: $f_\theta(z(t), t)$
- Deep Learning Researcher: Stack layers

Motivation $\dot{z} = z$

- Consider (some how) we train ODE: $\dot{z} = f_\theta(z, t)$
- How do we apply it?
  By (**numerically**) solving IVP problem: $z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_\theta(z(t), t)\, dt$
- Numerical Solver evaluates: $f_\theta(z(t), t)$
- Deep Learning Researcher: Stack layers

Contradiction

Idea

- ODE should have “simple dynamics”

Idea

- ODE should have “simple dynamics”

Simple vector field

Idea

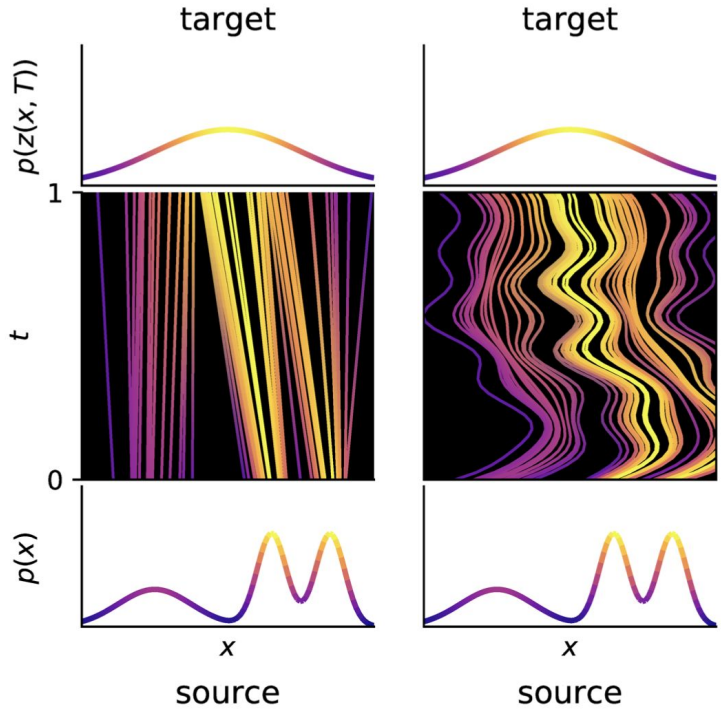
- ODE should have “simple dynamics”

Simple vector field

Not so simple

Idea

- ODE should have “simple dynamics”
- “How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization”

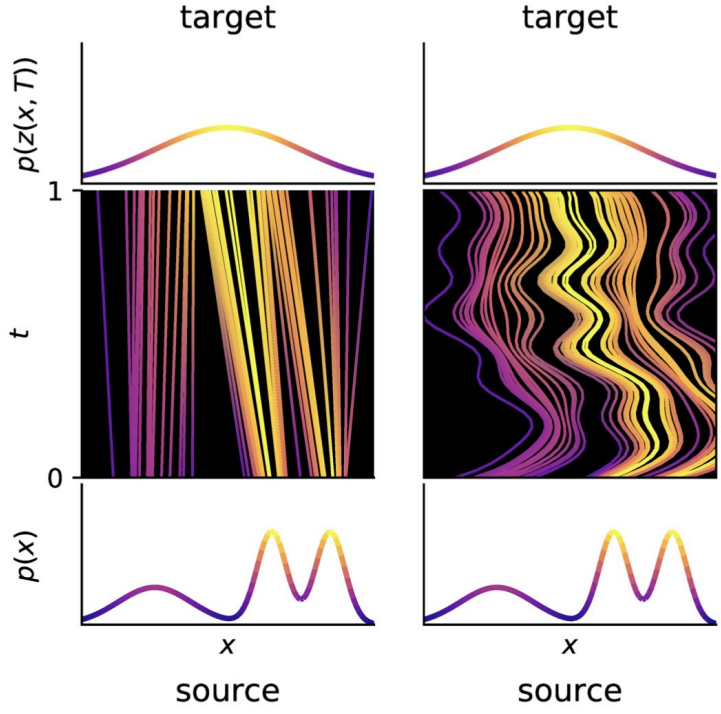


(a) Optimal transport map (b) generic flow

Chris Finlay [1] Jörn-Henrik Jacobsen [2] Levon Nurbekyan [3] Adam M Oberman [1]

Idea

- ODE should have “simple dynamics”
- [“How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization”]
- Straight lines are good
- Optimal transport formalization of “simple”



(a) Optimal transport map (b) generic flow

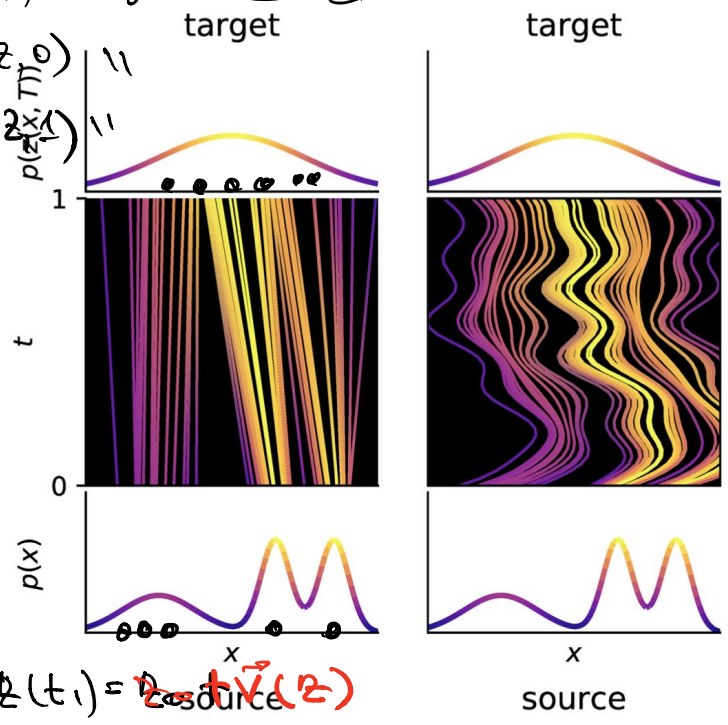
Chris Finlay [1] Jörn-Henrik Jacobsen [2] Levon Nurbekyan [3] Adam M Oberman [1]

Idea

[“How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization”]

- Straight lines are good
- Optimal transport formalization of “simple”
- Heavy math boils down to regularization:

$$\left[\|f(\mathbf{z}(t),t,\theta)\|_2^2 + \|\nabla_{\mathbf{z}} f(\mathbf{z}(t),t,\theta)\|_F^2\right] + h_{loss}$$



(a) Optimal transport map (b) generic flow

Chris Finlay [1] Jörn-Henrik Jacobsen [2] Levon Nurbekyan [3] Adam M Oberman [1]

Motivation

$$\|f(\mathbf{z}(t), t, \theta)\|_2^2$$

- Less evaluations = **better**
- What Influence on #Evaluations?
  (Number of Function Evaluation)

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_\theta(z(t), t)\, dt$$

$$f_\theta(z(t), t)$$

Motivation

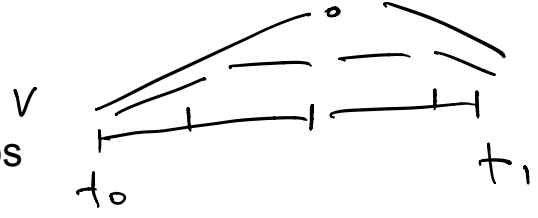
$\|f(\mathbf{z}(t), t, \theta)\|_2^2$

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_\theta(z(t), t)\, dt$$

$f_\theta(z(t), t)$

- Less evaluations = **better**
- What Influence on #Evaluations?
  (Number of Function Evaluation)

- Step size: bigger step, less number of steps



Motivation

$\|f(\mathbf{z}(t), t, \theta)\|_2^2$

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_\theta(z(t), t)\, dt$$

$f_\theta(z(t), t)$

- Less evaluations = **better**
- What Influence on #Evaluations?
  (Number of Function Evaluation)

- Step size: bigger step, less number of steps
- But local error should be bounded, so:
  - Higher order of method: more evaluations
  - Adaptive step size:
    - Rejection of trajectory leads for more evaluations

Motivation

$$\|f(\mathbf{z}(t), t, \theta)\|_2^2$$

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_\theta(z(t), t)\, dt$$

$f_\theta(z(t), t)$

$f_\theta(g_\theta(z(t_0), t_0))$

- Less evaluations = **better**
- What Influence on #Evaluations?
  (Number of Function Evaluation)

- Step size: bigger step, less number of steps
- But local error should be bounded, so:
  - Higher order of method: more evaluations
  - Adaptive step size:
    - Rejection of trajectory leads for more evaluations

Regularize properties of $f_\theta(z(t), t)$ **that adaptive step-size takes in account**

Subset of Runge-Kutta methods

- One step [всё будущее зависит от x_n]
- Multi-stage
- Explicit

$$x(t_{n+1}) = x_{n+1} = x_n + \Delta t \sum_{i=1}^{m} c_i k_i$$

slopes

$$k_1 = f(t_n, x_n)$$

$$k_2 = f(t_n + \alpha_2 \Delta t,\ x_n + \Delta t\, \beta_{21} k_1)$$

$$\vdots$$

$$k_m = f\left(t_n + \alpha_m \Delta t,\ x_n + \Delta t \sum_{j=1}^{m-1} \beta_{mj} k_j\right)$$

Example

- Let’s derive some method as it is easy:
  - Taylor Series of True Trajectory and match its order
  - * Hand derivations here *

Blank

$\dot{x} = f(x,t)$ → $\begin{bmatrix} \dot{x} \\ \dot{t} \end{bmatrix} = \begin{bmatrix} f([x,t]) \\ 1 \end{bmatrix}$ $\dot{x} = f(x)$

$x(t+\Delta t) = \Phi(x(t), \Delta t)$ → flow map, точноСть.

$x_{n+1} = \Psi(x_n, \Delta t)$ → numeric, not exact.

Local Error = $|x_{n+1} - x(t+\Delta t)| = \varepsilon_i$, Global = $\sum' \varepsilon_i$

time $|C|t^4$ $\lesssim f^{(4)}$

f. $\Phi(x(t), \Delta t) \Rightarrow$: $x(t) = x(0) + t\dot{x} + \frac{t^2}{2}\ddot{x} + \frac{t^3}{6}\dddot{x} + \ldots$

$\dot{x} = f(x)$, $\ddot{x} = \frac{d}{dt} f(x(t)) = f_x \dot{x} = f_x f$.

$f_{j,kl} = \frac{\partial f_j}{\partial x_k \partial x_l}$

$\dddot{x}_j = \frac{d}{dt}\ddot{x}_j = \frac{d}{dt}\left[\frac{\partial f_j}{\partial x_k} f_k\right] = \frac{d}{dt} f_{j,k} f_k =$

$= f_{j,kl} f_l f_k + f_{j,k} f_{k,l} f_l$.

$x_j(t) = x_j + t f_j + \frac{1}{2}t^2 f_{j,k} f_k + \frac{1}{6}t^3 [\;\;]$

Blank $\dot{x} = f(t, x)$ $x_{k+1} = x_k + \Delta t \, \Phi =$

$= x_k + \Delta t \left[A f(t_k, x_k) + B f\left(t_k + P\Delta t, \right.\right.$ через δ корней

$\left.\left. x_k + Q f(t_k, x_k) \Delta t \right) \right]$

$f(t+\Delta t, x+\Delta h) = f(t,x) + f_t(t,x)\Delta t + f_x(t,x)\Delta h$

$\frac{1}{2!}\ddot{x} = f_t + f_x f$

$x_{k+1} = x_k + \Delta t \, f(t_k, x_k)(A+B) + \frac{\Delta t^2}{2} B (P f_t + Q f_x f) + O(\Delta t^3)$

$\begin{cases} A+B=1 \\ BP = \frac{1}{2}, \; BQ = \frac{1}{2}. \end{cases}$

(v) идут в коэффициенты

RK45 $\frac{1}{2}$, $\frac{1}{3}$

R6 0.111738

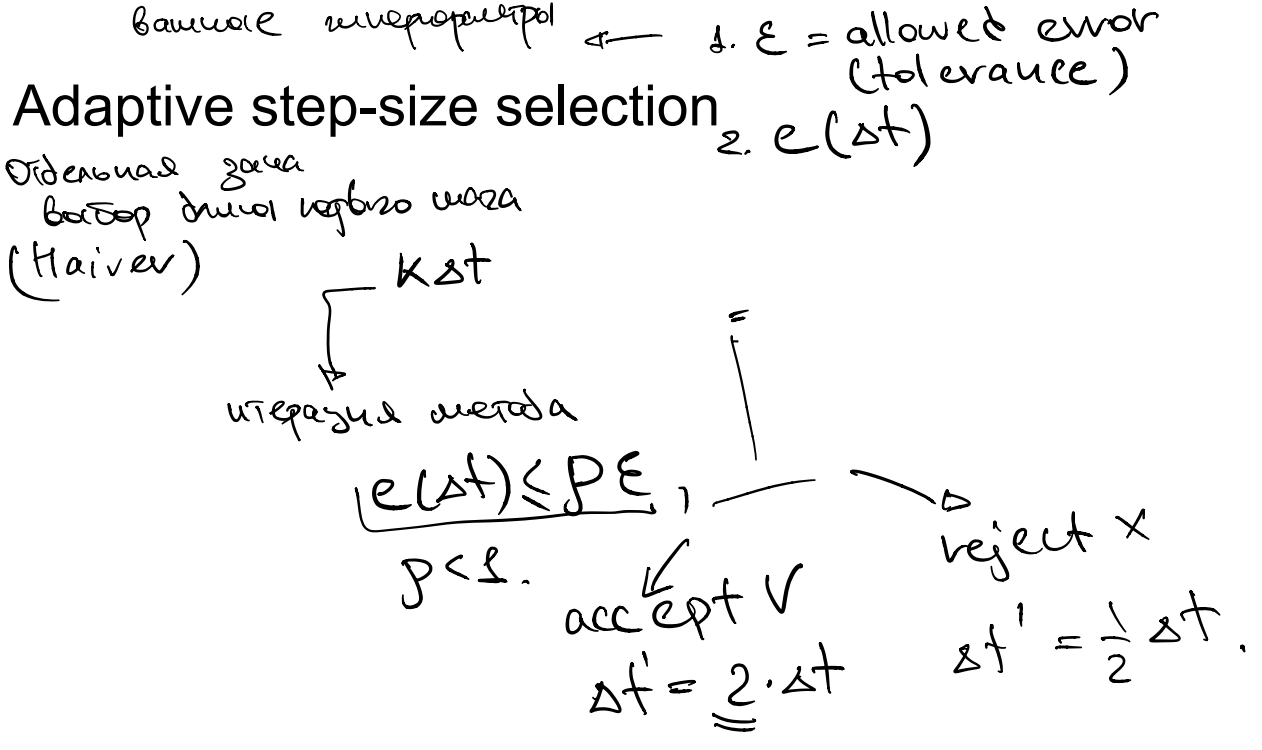
Example

- Let’s derive some method as it is easy:
  - Taylor Series of True Trajectory and match its order
  - * Hand derivations here *
- How to get higher orders in approximation?
  - Higher order -> we need to make t * t some where
  - Use different points to measure slope and average them
  - * Hand derivations here *

Blank

Blank

Adaptive step-size selection



Adaptive step-size selection

$x_{n+1} = x_{true} + Ch^4$

$\tilde{x}_{n+1} = x_{true} + Ch^5$

$|\tilde{x}_{n+1} - x_n| \sim (\Delta t)^5$ → используем

"ε" $= \frac{(\Delta t)^5}{\Delta T}$ → должно

ε

"Bootstrap"

$C \leq \sup \left| \frac{d^{k+1}}{dt^{k+1}} x \right|$

RK5 — в 5 точках посчитали f ~ столько-то свободных коэфф.

Embedded RK 4

Regularization

$$\mathcal{R}_K(\theta) = \int_{t_0}^{t_1} \left\| \frac{\mathrm{d}^K \mathbf{z}(t)}{\mathrm{d}t^K} \right\|_2^2 \mathrm{d}t$$

$\dot{z} = f(z, t)$

Assume, $\mathcal{R}_K(\theta) = 0$

- K = 0 X

$\int_{t_0}^{t'} \langle z_j, z_j \rangle dt = 0 \;,\; z_j(t) = 0$

Regularization

$$\mathcal{R}_K(\theta) = \int_{t_0}^{t_1} \left\| \frac{\mathrm{d}^K \mathbf{z}(t)}{\mathrm{d}t^K} \right\|_2^2 \mathrm{d}t$$

Assume, $\mathcal{R}_K(\theta) = 0$

- K = 0

$$\langle \dot{z}, \dot{z} \rangle = 0 \qquad \dot{z}_j = 0 \Rightarrow z_j = \text{const} \;\; \forall j$$

- K = 1

Regularization

$$\mathcal{R}_K(\theta) = \int_{t_0}^{t_1} \left\| \frac{\mathrm{d}^K \mathbf{z}(t)}{\mathrm{d}t^K} \right\|_2^2 \mathrm{d}t$$

Assume, $\mathcal{R}_K(\theta) = 0$

- K = 0

- K = 1

- K = 2

$$z(t) = z(0) + \frac{\dot{z}}{1} t + \frac{\ddot{z}}{2} t^2 + \dots$$

Experiments

R_3.

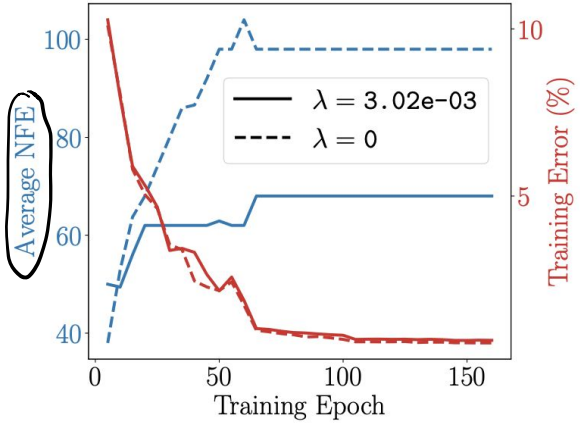


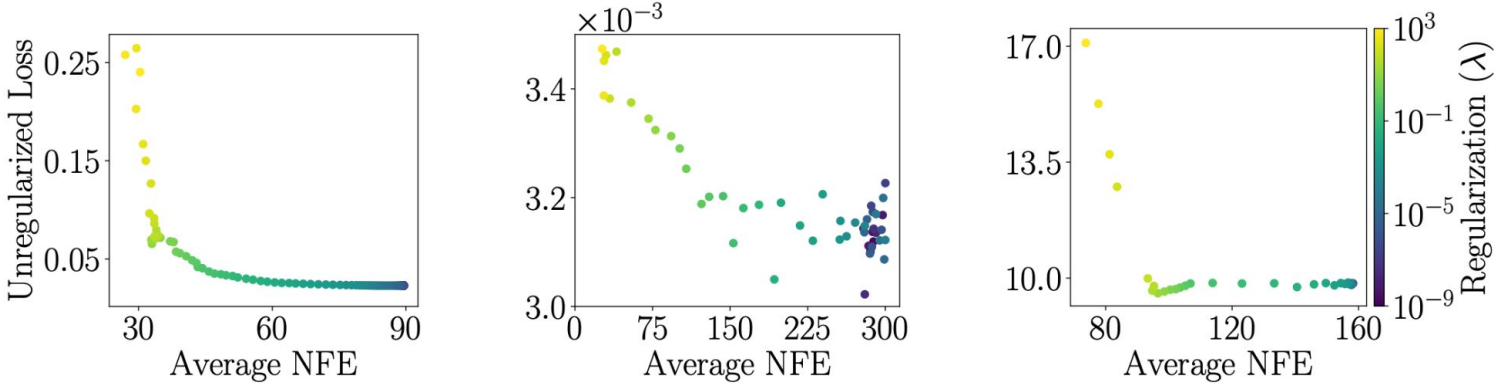
Figure 3: Number of function evaluations (NFE) and training error during training. Speed regularization (solid) decreases the NFE throughout training without substantially changing the training error.

training time ↑

5.1 Supervised Learning

We construct a model for MNIST classification: it takes in as input a flattened MNIST image and integrates it through dynamics given by a simple MLP, then applies a linear

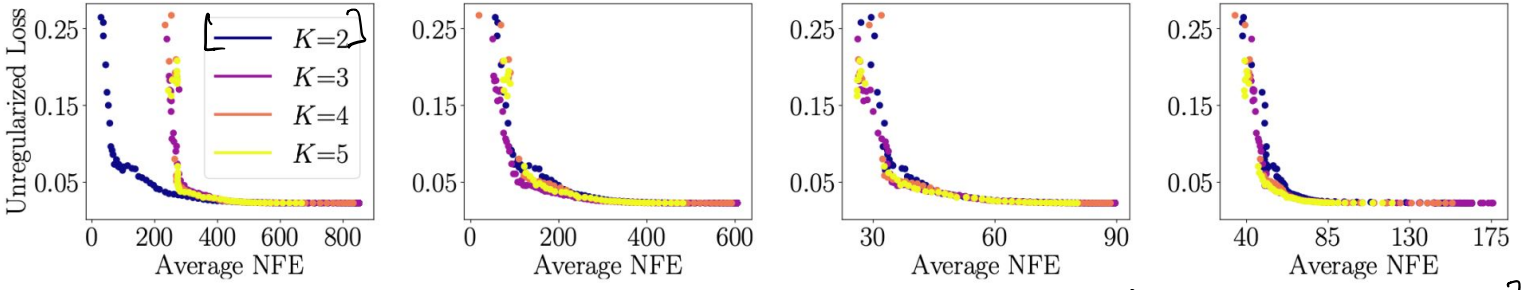
Experiments



(a) MNIST Classification (b) PhysioNet Time-Series (c) Miniboone Density Estimation

Figure 5: Tuning the regularization of $\mathcal{R}_2$ trades off between training loss and solver speed in three different applications of neural ODEs. Horizontal axes show average number of function evaluations, and vertical axes show unregularized training loss, both at the end of training.

controlling $\mathcal{R}_2$.



(a) Order 2 Solver (b) Order 3 Solver (c) Order 5 Solver (d) Adaptive Order Solver

Figure 6: Comparing tradeoff between speed and performance when regularizing different orders. 6a): For a 2nd-order solver, regularizing the 2nd total derivative gives the best tradeoff. 6b): For a 3rd-order solver, regularizing the 3rd total derivative gives the best tradeoff, but the difference is small. 6c): For a 5th-order solver, results are mixed. 6d): For an adaptive-order solver, the difference is again small but regularizing higher orders works slightly better.